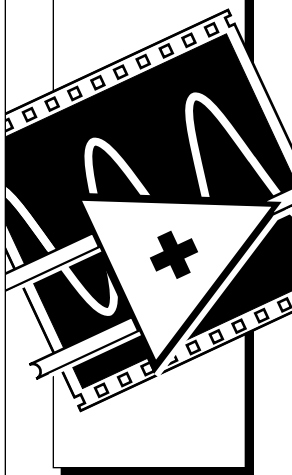


LabVIEW



LabVIEW[®] VXi VI Reference Manual

January 1996 Edition
Part Number 320557C-01



Internet Support

GPiB: gpib.support@natinst.com

DAQ: daq.support@natinst.com

VXI: vxi.support@natinst.com

LabVIEW: lv.support@natinst.com

LabWindows: lw.support@natinst.com

HiQ: hiq.support@natinst.com

E-mail: info@natinst.com

FTP Site: [ftp.natinst.com](ftp://ftp.natinst.com)

Web Address: <http://www.natinst.com>



Bulletin Board Support

BBS United States: (512) 794-5422 or (800) 327-3077

BBS United Kingdom: 01635 551422

BBS France: 1 48 65 15 59



FaxBack Support

(512) 418-1111 or (800) 329-7177



Telephone Support (U.S.)

Tel: (512) 795-8248

Fax: (512) 794-5678 or (800) 328-2203



International Offices

Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,

Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,

Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,

Italy 02 48301892, Japan 03 5472 2970, Korea 02 596 7456, Mexico 95 800 010 0793,

Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085,

Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

Except as specified herein, National Instruments makes no warranties, express or implied, and specifically disclaims any warranty of merchantability or fitness for a particular purpose. Customer's right to recover damages caused by fault or negligence on the part of National Instruments shall be limited to the amount theretofore paid by the customer. National Instruments will not be liable for damages resulting from loss of data, profits, use of products, or incidental or consequential damages, even if advised of the possibility thereof. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

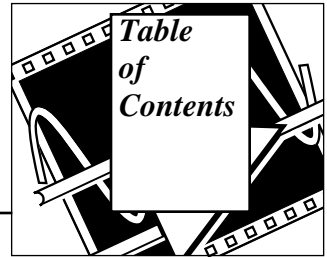
Trademarks

LabVIEW® and NI-VXI™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.



About This Manual

Organization of This Manual	xi
Conventions Used in This Manual	xii
Related Documentation	xiv
Customer Communication	xv

Chapter 1 Introduction

VXIbus Overview	1-1
VXI Devices	1-1
Register-Based Devices	1-2
Message-Based Devices	1-2
Word Serial Protocol	1-3
Commander/Servant Hierarchies	1-4
Interrupts and Asynchronous Events	1-4
VXI Handler VIs Overview	1-5
VXI VI Library Overview	1-6
Multiple Mainframe Support	1-8
Embedded Versus External and Extended Controllers	1-8
Extender Versus Controller Parameters	1-10

Chapter 2 System Configuration VIs

Locating System Configuration VIs in LabVIEW	2-1
Finding Help Online for System Configuration VIs	2-1
System Configuration VI Descriptions	2-2
CloseVXIlibrary	2-2
CreateDevInfo	2-2
FindDevLA	2-3
GetDevInfoLong	2-4
GetDevInfoShort	2-5
GetDevInfoStr	2-6
InitVXIlibrary	2-7
SetDevInfoLong	2-7

SetDevInfoShort	2-8
SetDevInfoStr	2-10

Chapter 3

Word Serial Commander Protocol VIs

Locating Word Serial Commander VIs in LabVIEW	3-1
Finding Help Online for Word Serial Commander VIs	3-1
Word Serial Commander VI Descriptions	3-2
WSabort	3-2
WSclr	3-3
WScmd	3-3
WSEcmd	3-5
WSgetTmo	3-6
WSLcmd	3-7
WSLresp	3-8
WSrd	3-9
WSresp	3-12
WSsetTmo	3-13
WStrg	3-13
WSwrt	3-15

Chapter 4

Word Serial Servant Protocol VIs

Locating Word Serial Servant Protocol VIs in LabVIEW	4-2
Finding Help Online for Word Serial Servant Protocol VIs	4-3
Word Serial Servant Protocol VI Descriptions	4-3
GenProtError	4-3
GetWSScmdHandler	4-4
GetWSSecmdHandler	4-4
GetWSSLcmdHandler	4-5
GetWSSrdHandler	4-5
GetWSSwrtHandler	4-5
PollWSScmdHandler	4-6
PollWSSecmdHandler	4-6
PollWSSLcmdHandler	4-7
PollWSSrdHandler	4-7
PollWSSwrtHandler	4-9
RespProtError	4-10
SetWSScmdHandler	4-10
SetWSSecmdHandler	4-11
SetWSSLcmdHandler	4-12
SetWSSrdHandler	4-12

SetWSSwrtHandler	4-13
WSSabort	4-13
WSSdisable	4-14
WSSenable	4-14
WSSLnoResp	4-15
WSSLsendResp	4-15
WSSnoResp	4-16
WSSrd	4-16
WSSsendResp	4-17
WSSwrt	4-18

Chapter 5

Low-Level VXIbus Access VIs

Multiple Pointer Access for a Window	5-2
Owner Privilege	5-2
Access Only Privilege	5-3
Locating Low-Level VXIbus Access VIs in LabVIEW	5-4
Finding Help Online for Low-Level VXIbus Access VIs	5-4
Low-Level VXIbus Access VI Descriptions	5-4
GetByteOrder	5-4
GetContext	5-5
GetPrivilege	5-5
GetVXIbusStatus	5-6
GetVXIbusStatusInd	5-7
GetWindowRange	5-8
MapVXIAddress	5-9
SetByteOrder	5-10
SetContext	5-11
SetPrivilege	5-12
UnMapVXIAddress	5-12
VXIpeek	5-13
VXIpoke	5-14

Chapter 6

High-Level VXIbus Access VIs

Locating High-Level VXIbus Access VIs in LabVIEW	6-2
Finding Help Online for High-Level VXIbus Access VIs	6-2
High-Level VXIbus Access VI Descriptions	6-2
VXIin	6-2
VXIinReg	6-4
VXImove	6-5

VXIout	6-8
VXIoutReg	6-9

Chapter 7

Local Resource Access VIs

Locating Local Resource Access VIs in LabVIEW	7-1
Finding Help Online for Local Resource Access VIs	7-1
GetMyLA	7-2
ReadMODID	7-2
SetMODID	7-3
VXIinLR	7-3
VXImemAlloc	7-4
VXImemCopy	7-5
VXImemFree	7-6
VXIoutLR	7-7

Chapter 8

VXI Signal VIs

Locating VXI Signal VIs in LabVIEW	8-2
Finding Help Online for VXI Signal VIs	8-2
VXI Signal VI Descriptions	8-2
DisableSignalInt	8-2
EnableSignalInt	8-3
GetSignalHandler	8-3
PollSignalHandler	8-4
RouteSignal	8-4
SetSignalHandler	8-6
SignalDeq	8-7
SignalEnq	8-9
SignalJam	8-9
WaitForSignal	8-10

Chapter 9

VXI Interrupt VIs

Locating VXI Interrupt VIs in LabVIEW	9-2
Finding Help Online for VXI Interrupt VIs	9-2
VXI Interrupt VI Descriptions	9-2
AcknowledgeVXIint	9-2
AssertVXIint	9-3
DeAssertVXIint	9-4
DisableVXIint	9-4

DisableVXItoSignalInt	9-5
EnableVXIint	9-6
EnableVXItoSignalInt	9-6
GetVXIintHandler	9-7
PollVXIintHandler	9-8
RouteVXIint	9-8
SetVXIintHandler	9-9
VXIintAcknowledgeMode	9-10

Chapter 10

VXI Trigger VIs

Locating VXI Trigger VIs in LabVIEW	10-1
Finding Help Online for VXI Trigger VIs	10-1
VXI Trigger VI Descriptions	10-2
AcknowledgeTrig	10-2
DisableTrigSense	10-3
EnableTrigSense	10-4
GetTrigHandler	10-5
MapTrigToTrig	10-6
PollTrigHandler	10-8
SetTrigHandler	10-10
SrcTrig	10-11
TrigAssertConfig	10-12
TrigCntrConfig	10-13
TrigExtConfig	10-15
TrigTickConfig	10-16
UnMapTrigToTrig	10-18
WaitForTrig	10-19

Chapter 11

System Interrupt Handler VIs

Locating System Interrupt Handler VIs in LabVIEW	11-1
Finding Help Online for System Interrupt Handler VIs	11-1
System Interrupt Handler VI Descriptions	11-2
AssertSysreset	11-2
DisableACfail	11-3
DisableSoftReset	11-3
DisableSysfail	11-4
DisableSysreset	11-4
EnableACfail	11-5
EnableSoftReset	11-5
EnableSysfail	11-6

EnableSysreset	11-6
GetACfailHandler	11-7
GetBusErrorHandler	11-7
GetSoftResetHandler	11-7
GetSysfailHandler	11-8
GetSysresetHandler	11-8
PollACfailHandler	11-9
PollBusErrorHandler	11-9
PollSoftResetHandler	11-10
PollSysfailHandler	11-10
PollSysresetHandler	11-11
SetACfailHandler	11-11
SetBusErrorHandler	11-12
SetSoftResetHandler	11-12
SetSysfailHandler	11-13
SetSysresetHandler	11-13

Chapter 12

VXIbus Extender VIs

Locating VXIbus Extender VIs in LabVIEW	12-1
Finding Help Online for VXIbus Extender VIs	12-1
VXIbus Extender VI Descriptions	12-2
MapECLtrig	12-2
MapTTLtrig	12-3
MapUtilBus	12-4
MapVXIint	12-5

Chapter 13

VXIbus Supplemental VIs

Locating VXIbus Supplemental VIs in LabVIEW	13-1
Finding Help Online for VXIbus Supplemental VIs	13-1
VXIbus Supplemental VI Descriptions	13-2
PollACfailHandlerTmo	13-2
PollBusErrorHandlerTmo	13-2
PollSignalHandlerTmo	13-3
PollSoftResetHandlerTmo	13-3
PollSysfailHandlerTmo	13-4
PollSysresetHandlerTmo	13-4
PollTrigHandlerTmo	13-5
PollVXIintHandlerTmo	13-6
PollWSScmdHandlerTmo	13-7
PollWSSEcmdHandlerTmo	13-8

PollWSSLcmdHandlerTmo	13-8
PollWSSrdHandlerTmo	13-9
PollWSSwrtHandlerTmo	13-10

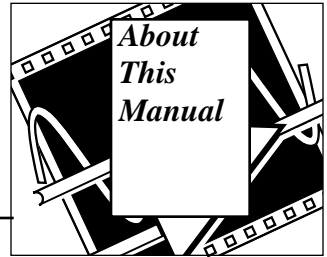
Appendix

Customer Communication

Figures

Figure 1-1. VXI Configuration Registers	1-2
Figure 1-2. VXI Software Protocols	1-3
Figure 1-3. Example of an Interrupt Handler	1-6
Figure 1-4. Embedded Versus External CPU Configurations	1-9
Figure 1-5. Extender Versus Controller Parameters	1-10

Table of Contents



The *LabVIEW VXI VI Reference Manual* describes the VXI virtual instruments (VIs) for LabVIEW.

This manual supplements your LabVIEW user manual and assumes that you are familiar with that material. You should also be familiar with the operation of LabVIEW, your computer, and your computer's operating system.

Organization of This Manual

The *LabVIEW VXI VI Reference Manual* is organized as follows:

- Chapter 1, *Introduction*, presents an overview of VXI concepts and the VXI virtual instrument (VI) library.
- Chapter 2, *System Configuration VIs*, describes the System Configuration VIs.
- Chapter 3, *Word Serial Commander Protocol VIs*, describes the VXI Word Serial Commander Protocol VIs.
- Chapter 4, *Word Serial Servant Protocol VIs*, describes the VXI Word Serial Servant Protocol VIs.
- Chapter 5, *Low-Level VXIbus Access VIs*, describes the use of the VIs that give you the fastest access method for directly reading from or writing to any of the VXIbus address spaces.
- Chapter 6, *High-Level VXIbus Access VIs*, describes the VIs with which you have direct access to the VXIbus address spaces.
- Chapter 7, *Local Resource Access VIs*, describes VIs you use to access miscellaneous local resources such as the local CPU VXI register set, Slot 0 MODID operations, and the local CPU VXI Shared RAM.
- Chapter 8, *VXI Signal VIs*, describes the VIs you use to specify signal routing, manipulate the global signal queue, and wait for a particular VXI signal to be received.

- Chapter 9, *VXI Interrupt VIs*, describes the VIs that control VXI interrupts. VXI interrupts are a basic form of asynchronous communication used by VXI devices with VXI interrupter support.
- Chapter 10, *VXI Trigger VIs*, describes the VIs that control triggers, a backplane feature that VXI added to the VME standard.
- Chapter 11, *System Interrupt Handler VIs*, describes the System Interrupt Handler VIs.
- Chapter 12, *VXIbus Extender VIs*, describes the VXIbus Extender VIs.
- Chapter 13, *VXIbus Supplemental VIs*, describes the VXIbus Supplemental VIs, which have a **timeout** input that allows you to stop waiting if the interrupt condition for which you are waiting does not occur.
- The appendix, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list of terms used in this manual and a description of each.
- The *Index* contains an alphabetical list of key terms and topics used in this manual, including the page where each one can be found.

Conventions Used in This Manual

The following conventions are used in this manual.

bold	Bold text denotes menus, menu items, or dialog box buttons or options. In addition, bold text denotes VI input and output parameters. In the Help window pictures of VI inputs and outputs, boldfaced parameters are parameters whose values you must specify.
<i>italic</i>	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. In this manual, italics are also used to denote Word Serial commands and queries.
<i>bold italic</i>	Bold italic text denotes a note, caution, or warning.
<>	Angle brackets enclose the name of a key on the keyboard—for example, <shift>.













-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Ctrl-Alt-Delete>.
<enter>	Key names are lowercase.
IEEE 488.1 and IEEE 488.2	IEEE 488.1 and IEEE 488.2 refer to the ANSI/IEEE Standard 488.1-1987 and the ANSI/IEEE Standard 488.2-1987, respectively, which define the GPIB.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence File»Page Setup»Options»Substitute Fonts directs you to pull down the File menu, select the Page Setup item, select Options , and finally select the Substitute Fonts option from the last dialog box.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in <code>drivename\dir1name\dir2name\myfile</code> .



Note: *This icon to the left of bold italicized text denotes a note, which alerts you to important information.*

Each VXI VI description in this manual displays an icon before the parameter name to designate its data type. These icons are illustrated and defined in the following table.

Control	Indicator	Data Type
		Integer
		Long Integer
		Unsigned Integer
		Unsigned Long Integer
		String
		Boolean

Control	Indicator	Data Type
		Array of Unsigned Long Integer
		Array of Unsigned Integer
		Array of Unsigned Character
		Array of Integer
		Array of Boolean
		Cluster

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

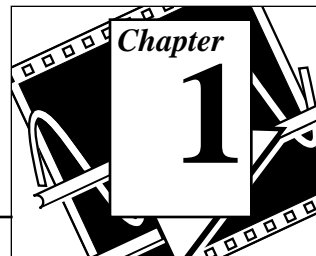
The following documents contain information that you may find helpful as you read this manual:

- The getting started or user manuals for the VXI boards you use
- Your *LabVIEW Tutorial Manual*
- *IEEE Standard for a Versatile Backplane Bus: VMEbus*, ANSI/IEEE Standard 1014-1987
- VXI-1, *VXIbus System Specification*, Rev. 1.4, VXIbus Consortium (available from National Instruments Corporation)
- VXI-6, *VXIbus Mainframe Extender Specification*, Rev. 1.0, VXIbus Consortium (available from National Instruments Corporation)
- *NI-VXI Software Reference Manual for C* (available from National Instruments Corporation)

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in the appendix, *Customer Communication*, at the end of this manual.

Introduction



This manual is a companion guide to the *NI-VXI Software Reference Manual for C* that came with your VXI hardware. With the exception of the LabVIEW Poll Handler VIs, which correspond to the SetHandler and DefaultHandler functions, every VI in this manual corresponds directly with the function of the same name in the *NI-VXI Software Reference Manual for C*.

National Instruments recommends that you begin by reading Chapters 1 and 2 of the *NI-VXI Software Reference Manual for C*, as well as this chapter, to give you an overview of the VXIbus and NI-VXI. Once you are familiar with this material, you can begin to write your program. Notice that the beginning of each function chapter in both the *NI-VXI Software Reference Manual for C* and this book contain a general overview of the operations of the class of functions in the chapter, as well as a brief description of each function. You should become familiar with the entire class of functions you are using, so that you know which ones are necessary to perform your tasks most efficiently.

Remember to always check the README.DOC file in your NI-VXI directory to obtain the latest information about your software, as well as platform specific information regarding your hardware.

VXIbus Overview

This section introduces some of the concepts from the VXIbus specification.

VXI Devices

A VXI device has a unique logical address, which you use to find or access the device in the VXI system. This logical address can be compared to a GPIB **device** address. Because VXI uses an 8-bit logical address, you can have up to 256 VXI devices in a VXI system.

Each VXI device must have a specific set of registers, called *configuration registers*. These registers are located in the upper 16 KB of the 64-KB A16 VXI address space, as shown in Figure 1-1. The

logical address of a VXI device determines the location of the configuration registers of the device in the 16-KB area reserved by VXI.

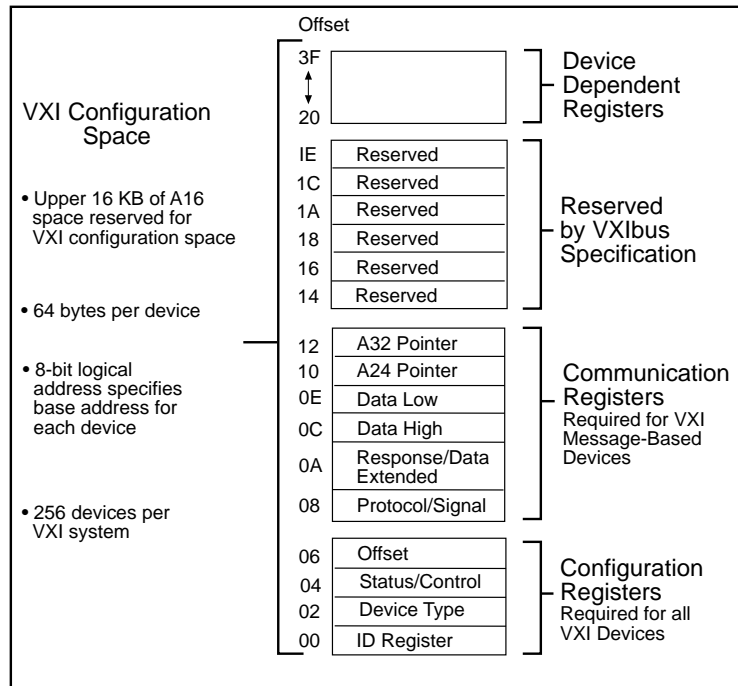


Figure 1-1. VXI Configuration Registers

Register-Based Devices

VXI configuration registers, which are required for all VXI devices, help the system identify each VXI device, its type, model and manufacturer, address space, and memory requirements. VXIbus devices with only this minimum level of capability are called *Register-Based* devices. With this common set of configuration registers, the centralized Resource Manager (RM), which is essentially a software module, can perform automatic system and memory configuration when the system is initialized.

Message-Based Devices

In addition to Register-Based devices, the VXIbus specification also defines *Message-Based* devices, which must have *communication registers* in addition to the configuration registers. All Message-Based

VXIbus devices, regardless of the manufacturer, can communicate at a minimum level using the VXI-specified Word Serial Protocol, as shown in Figure 1-2. At this minimum level, you can establish higher-performance communication channels, such as shared-memory channels to take advantage of the VXIbus bandwidth capabilities.

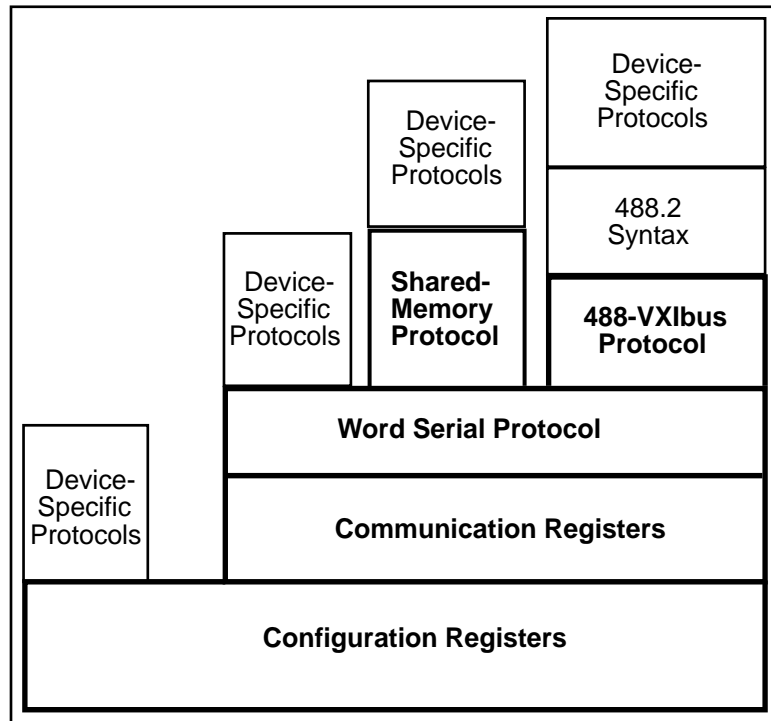


Figure 1-2. VXI Software Protocols

Word Serial Protocol

The VXIbus Word Serial Protocol is a standardized message-passing protocol. This protocol functions much like the IEEE 488 protocol, which transfers data messages to and from devices one byte (or word) at a time. Thus, VXI Message-Based devices communicate in a fashion very similar to IEEE 488 instruments. In general, Message-Based devices typically contain some level of local intelligence that uses or requires a high level of communication.

All VXI Message-Based devices must use Word Serial Protocol and communicate in a standard way. The protocol is called *word serial*, because if you want to communicate with a Message-Based device, you do so by writing and reading 16-bit words one at a time to and from the Data In (write Data Low) and Data Out (read Data Low) hardware registers located on the device itself. Word serial communication is paced by bits in the response register of the device that indicate whether the Data In register is empty and whether the Data Out register is full.



Note: *In this manual, italics are also used to denote Word Serial commands and queries.*

Commander/Servant Hierarchies

The VXIbus specification defines a Commander/Servant communication protocol so that you can construct hierarchical systems using conceptual layers of VXI devices. This structure could be compared to an inverted tree. A *Commander* is any device in the hierarchy with one or more associated lower level devices, or Servants. A *Servant* is any device in the subtree of a Commander. A device can be both a Commander and a Servant in a multiple-level hierarchy.

A Commander has exclusive control of the communication and configuration register of its immediate Servants (one or more). Any VXI module has only one Commander. Commanders communicate with Servants through the communication registers of the Servants using the Word Serial Protocol. Servants communicate with their Commander, responding to the Word Serial commands and queries from their Commander through the Word Serial protocol. Servants can also communicate asynchronous status and events to their Commander through hardware interrupts, or by writing specific signals directly to the Signal register of their Commander.

Interrupts and Asynchronous Events

Servants can communicate asynchronous status and events to their Commander through hardware interrupts or by writing specific messages (signals) directly to the hardware Signal register of their Commander. Devices that are not bus masters always transmit such information through interrupts, whereas devices that have bus master capability can either use interrupts or send signals. Some devices can receive only signals, but others might be only interrupt handlers.

The VXIbus specification contains defined Word Serial commands so that a Commander can understand the capabilities of its Servants and configure them to generate interrupts or signals in a particular way. For example, a Commander can instruct its Servants to use a particular interrupt line, send signals rather than generate interrupts, or configure the reporting of only certain status or error conditions.

Although the Word Serial Protocol is reserved for Commander/Servant communications, two VXI devices can establish peer-to-peer communication through a specified shared-memory protocol or by simply writing specific messages directly to the Signal register of the device.

VXI Handler VIs Overview

A VXI handler is a user routine that is executed when some event occurs in the VXI interface. For example, you can set a VXI interrupt handler to execute whenever an interrupt is asserted.

You can use VXI VIs to create a handler as part of your LabVIEW diagram. The handler can use any LabVIEW VI, including other VXI VIs. The VIs that you use to create these handlers are called the *Poll Handler VIs*.

The Poll Handler VIs are analogous to the SetHandler and DefaultHandler calls in the *NI-VXI Software Reference Manual for C*. The inputs to the Poll Handler VIs are the same as the parameters to the SetHandler functions. The outputs to the Poll Handler VIs are the same as the parameters to the DefaultHandler functions.

The Poll Handler VIs are implemented using LabVIEW occurrences. Because an occurrence is a *one-shot* event (the dependent VIs are executed only once), you must put the Poll Handler VI and your handler in a while loop structure.

Figure 1-3 shows an example of a LabVIEW interrupt handler, which displays the Dialog box whenever an interrupt of a certain level is asserted. The VXI VIs store the pending interrupts so that the handler executes once for each handler.

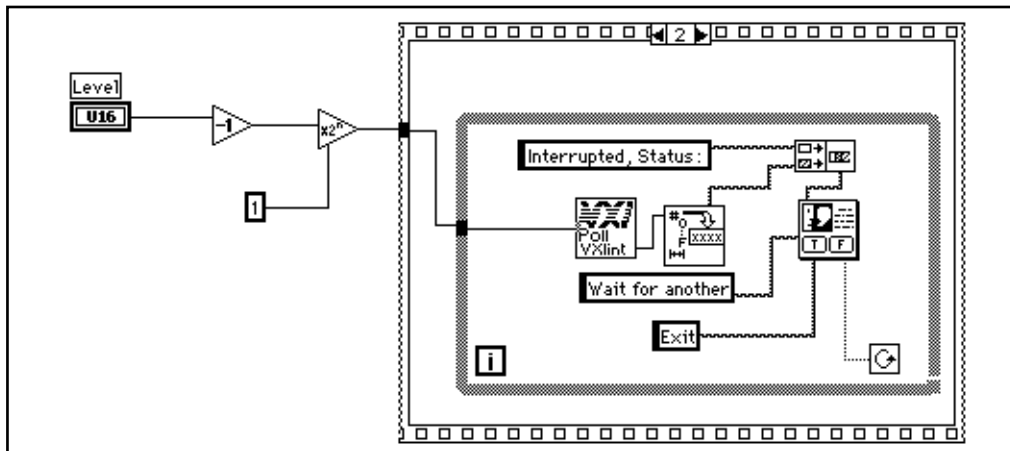


Figure 1-3. Example of an Interrupt Handler

Passing the value 3 to a Set Handler VI tells the VXI VI library to begin queuing occurrences for later handling by a Poll Handler VI.



Note: *Even on platforms where VXI user handlers are normally executed at interrupt time (such as Macintosh and Windows), the LabVIEW user handler diagram is not executed at interrupt time.*

VXI VI Library Overview

The *NI-VXI Software Reference Manual for C* gives an in-depth explanation of the NI-VXI functions and how they relate to the VXIbus specification. The material found in that manual applies equally to the NI-VXI LabVIEW VXI VIs, and you can use it as a reference in conjunction with this document. The VIs are divided into the following groups.

- *System Configuration VIs* initialize the NI-VXI interface at the lowest level. In addition, the System Configuration VIs can retrieve or modify device configuration information.
- *Word Serial Commander Protocol VIs.* Word Serial is the minimal mode of communication between VXI Message-Based devices. Word Serial Commanders can use these VIs to communicate with a Message-Based Servant device using the Word Serial, Longword Serial, or Extended Longword Serial protocols. These VIs can perform command/query sending and buffer reads/writes.

- *Word Serial Servant Protocol VIs* give Message-Based Servant devices all of the necessary capabilities to communicate with the Message-Based Commander of the local CPU (the device on which the NI-VXI interface resides) using the Word Serial, Longword Serial, or Extended Longword Serial protocols. These capabilities include command/query handling and buffer read/writes.
- *Low-Level VXIbus Access VIs*. Low-Level VXIbus access is the fastest access method for directly reading from or writing to any of the VXIbus address spaces. Use them when execution speed is critical.
- *High-Level VXIbus Access VIs* are similar to the Low-Level VXIbus Access VIs, but these VIs give you direct access to the VXIbus address spaces. You can use these VIs to read, write, and move blocks of data between any of the VXIbus address spaces. You can specify any VXIbus privilege mode or byte order. The VIs trap and report Bus Errors. The High-Level VXIbus Access VIs are easy-to-use. You should use them when execution speed is not crucial.
- *Local Resource Access VIs* offer access to miscellaneous local resources such as the VXI register set of local CPU, Slot 0 MODID operations (when the local device is configured for Slot 0 operation), and the VXI shared RAM of the local CPU. These VIs are useful for shared memory type communication, for non-Resource Manager operation (when the local CPU is not the Resource Manager), and for debugging purposes.
- *VXI Signal VIs* are used by VXI bus masters to interrupt another device. The value written to a Signal register of a device has the same format as the status/ID value returned during a VXI interrupt acknowledge cycle. You can route VXI signals to the default interrupt service routine or queue them on a global signal queue. VXI signal VIs can specify the signal routing, manipulate the global signal queue, and wait for a particular signal value (or set of values) to be received.
- *VXI Interrupt VIs* let you process individual VXI interrupt status/IDs as VME status/IDs, VXI status/IDs, or VXI signals. By default, status/IDs are processed as VXI signals. VXI interrupt VIs can specify the status/ID processing method and can assert specified VXI interrupt lines with a specified status/ID value.
- *VXI Trigger VIs* are a standard interface to source and accept any of the VXIbus TTL or ECL trigger lines. These VIs can also detect

acknowledgements from the accepting device and send the acknowledgements back to the sourcing device. You can use these VIs as configuration tools for signal conditioning and routing trigger lines, and for configuring the settings of the trigger inputs and outputs as well as the National Instruments Trigger Interface Chip (TIC) counter and tick timers. VXI trigger VIs support all VXI-defined trigger protocols; the actual capabilities depend on the specific hardware platform.

- *System Interrupt Handler VIs* let you handle the system interrupt conditions. These conditions include Sysfail, ACfail, Sysreset, BusError, and SoftReset interrupts.
- *VXibus Extender VIs* allow you to dynamically reconfigure multiple-mainframe transparent mapping of the VXI interrupt lines, TTL triggers, ECL triggers, and utility bus signals. The National Instruments Resource Manager configures the mainframe extenders with settings based on user-modifiable configuration files.

Multiple Mainframe Support

The VXI VIs described in this manual fully support multiple mainframes. The **mainframe** parameter in the VI denotes the logical address of the mainframe extender in the particular frame or the logical address of the embedded controller. The Startup Resource Manager supports one or more mainframe extenders and configures a single- or multiple-mainframe VXI system. Refer to VXI document VXI-6, *VXibus Mainframe Extender Specification*, Revision 1.0 for more details on multiple mainframes.

Embedded Versus External and Extended Controllers

The two basic types of multiple-mainframe configurations are the *embedded* CPU (controller) configuration and the *external* CPU (controller) configuration. The embedded CPU configuration is an intelligent CPU interface directly plugged into the VXI backplane. The embedded CPU must have all of its required VXI interface capabilities built onto the embedded CPU itself. An embedded CPU has direct access to the VXibus backplane for which it is installed. Access to other mainframes is done through the use of mainframe extenders.

The external CPU configuration involves plugging an interface device into an existing computer that connects the external CPU to VXI mainframes through one or more VXIbus *extended controllers*. An extended controller is a mainframe extender with additional VXIbus Controller capabilities.

Figure 1-4 illustrates the embedded and external CPU configurations.

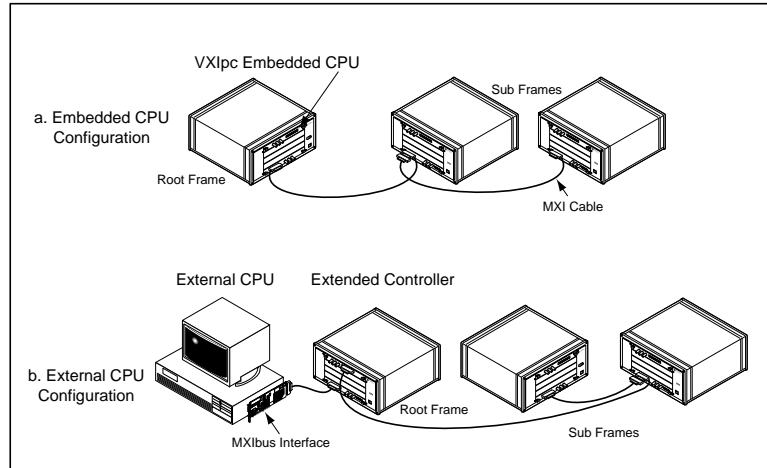


Figure 1-4. Embedded Versus External CPU Configurations

National Instruments MXIbus mainframe extender products include special features outside of the scope of the *VXIbus Mainframe Extender Specification* for more complete support of the VXIbus capabilities. These features give the external CPU all of the features of an embedded CPU, including VXI interrupt, TTL trigger, ECL trigger, Sysfail, ACfail, and Sysreset support for VXI systems. The external computer uses these features to interrupt on, sense, and/or assert these backplane signals. The capabilities of the MXIbus mainframe extender depend on the specific product and configuration.

Extended controllers exist only on the first level of mainframe hierarchy, as Figure 1-4 illustrates. The first level of hierarchy for the embedded CPU is always the local mainframe. This means the embedded CPU never has any extended controllers. An external CPU along with an extended controller is functionally equivalent to an embedded CPU configuration. An external CPU with more than one extended controller is a superset of the embedded CPU configuration.

If the application requires the local CPU (external or embedded) to receive VXI interrupts, triggers, and utility signals from below the first level of mainframe hierarchy, you should extend these VXIbus signals using the transparent VXIbus extender method (requiring INTX support on MXI extender products) via the Resource Manager configuration or VXIbus Extender functions.

Extender Versus Controller Parameters

This document uses the **extender** and **controller** parameters to specify the VXI mainframe to which a particular function applies. In general, the value of the **extender** or **controller** parameter is either the local CPU or the logical address of the VXI mainframe extender device that is used to access the particular mainframe (for example, a VXI-MXI or VME-MXI). Figure 1-5 shows an example of mainframe extenders used with the **extender** and/or **controller** parameters.

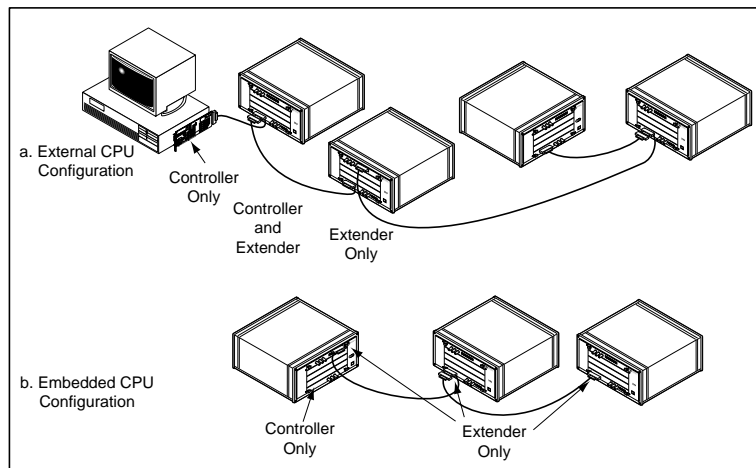


Figure 1-5. Extender Versus Controller Parameters

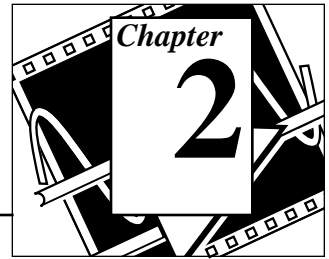
You can use the **extender** parameter only with the VXIbus Extender functions, which are fully described in Chapter 12, *VXIbus Extender VIs*. With these functions, you can reconfigure the transparent mainframe extension configured by the Resource Manager. The extensions included are VXI interrupts, TTL and ECL triggers, and Utility Bus (Sysfail, ACfail, and Sysreset). The capabilities of the VXIbus Extender functions are mapped directly onto the capabilities of the individual mapping registers of the standard VXIbus mainframe

extender. Because the Resource Manager configures the mainframe extenders with settings based on user-modifiable configuration files, your application probably will never need the VXIbus Extender functions.

The **controller** parameter appear only in NI-VXI functions that apply to embedded or extended controller capabilities. These capabilities include VXI interrupt, ACfail, Sysfail, and TTL/ECL trigger services. In embedded CPU configurations, you must always use a -1 or local CPU logical address for the **controller** parameter to specify the local resources of the embedded CPU. For external CPU configurations, a -1 or local CPU logical address specifies the first extended controller (mainframe extender with the lowest logical address).

You can use other values in external CPU configurations that have more than one extended controller. In this case, the **controller** parameter value specifies the particular extended controller for which the functions should apply. As a result, you can use different sets of VXIbus resources within individual first-level mainframes (for example, different interrupt levels handled on a per-mainframe basis). Notice that having more than one extended controller is not directly portable to the embedded CPU configuration.

System Configuration VIs



This chapter describes the System Configuration VIs. Your application program can use these VIs to copy all of the Resource Manager (RM) table information into data structures at startup so that you can find device names or logical addresses by specifying certain attributes of the device for identification purposes.

Locating System Configuration VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»System Configuration** to locate the System Configuration VIs in LabVIEW.

Finding Help Online for System Configuration VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

System Configuration VI Descriptions

CloseVXIlibrary

Disables interrupts and frees dynamic memory allocated for the internal device information table. You should call this VI before exiting your application.



status.

- 0: VXI library closed successfully.
- 1: Successful; previous InitVXIlibrary calls still pending.
- 1: VXI library is not open.

CreateDevInfo

Allocates space in the device information table for a new entry with logical address **la**. The fields in the device information table for the entry are set to default values (null or unasserted values).



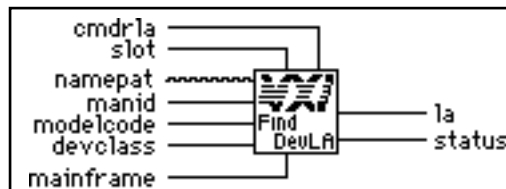
la is the logical address of the device for which to create an entry in the device information table.

I16**status.**

- 0: Entry successfully created.
- 1: **la** already exists.
- 2: **la** out of range 0 to 511.
- 3: Dynamic memory allocation failure.

FindDevLA

Finds a VXI device with the specified attributes in the RM table and returns its logical address. You can use any combination of attributes to specify a device. In this manner, you can acquire unknown device names or logical addresses. If **namepat** is "" or any other attribute is -1, or not connected, that attribute is not used in the matching algorithm. If two or more devices match, **la** contains the logical address of the first device found.

**I16****cmdr1a** is the logical address of the Commander.**I16****slot** is the slot location of the device.**abc****namepat** is the name pattern. A partial name is acceptable (for example, for GPIB-VXI, it will accept GP).**I16****manid** is the VXI manufacturer identification number.**U16****modelcode** is the 12-bit or 16-bit model number of the VXI manufacturer.**I16****devclass** is the device class of the device.

- 0: Memory Class Device.
- 1: Extended Class Device.
- 2: Message-Based Device.
- 3: Register-Based Device.

I16 **mainframe** is the mainframe location of the device (logical address of mainframe extender).

I16 **la** is the logical address of the device found.

I16 **status**.

- 0: A device matching the specification was found.
 - 1: No device matching the specification was found.
-

GetDevInfoLong

Gets information about a specified device from the device information table. The information is contained in a 32-bit unsigned integer.



I16 **la** is the logical address of the device to get information about.

U16 **field** is the field identification number.

- 12: Base of A24/A32 memory.
- 13: Size of A24/A32 memory.

U32 **longvalue** is the information for that field.

I16 **status**.

- 0: The specified information was returned.
 - 1: Device not found.
 - 2: Invalid **field** specified.
-

GetDevInfoShort

Gets information about a specified device from the device information table. The information is contained in a 16-bit unsigned integer.



I16

la is the logical address of the device to get information about.

U16

field is the field identification number.

- 2: Logical address of the Commander.
- 3: Mainframe.
- 4: Slot.
- 5: Manufacturer identification number.
- 7: Model code.
- 9: Device class.
- 10: Extended subclass (if extended class device).
- 11: Address space used.
- 14: Memory type and access time.
- 15: Bit vector list of VXI interrupter lines.
- 16: Bit vector list of VXI interrupt handler lines.
- 17: Mainframe extender, controller information.

Bits	Description
15 to 13	Reserved
12	1: Child side extender 0: Parent side extender
11	1: Frame extender 0: Not frame extender
10	1: Extended controller
9	1: Embedded controller
8	1: External controller
7 to 0	Frame extender towards root frame

- 18: Asynchronous mode control state.
- 19: Response enable state.
- 20: Protocols supported.
- 21: Capability/status flags.
- 22: Status state (Pass/Fail, Ready/Not Ready).

U16

shortvalue is the information for that field.

I16

status.

- 0: The specified information was returned.
- 1: Device not found.
- 2: Invalid **field** specified.

GetDevInfoStr

Gets information about a specified device from the device information table. The information is contained in **stringvalue**.



I16

la is the logical address of the device to get information about.

U16

field is the field identification number.

- 1: Device name.
- 6: Manufacturer name.
- 8: Model name.

abc

stringvalue is the information for that field.

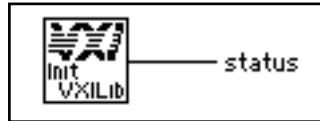
I16

status.

- 0: The specified information was returned.
- 1: Device not found.
- 2: Invalid **field** specified.

InitVXIlibrary

Allocates and initializes the data structures required by the VXI library VIs in the driver. This VI reads the RM table file and copies all of the RM information into data structures in local memory. It also performs other initialization operations, such as installing the default interrupt handlers.



I16 status.

- 0: VXI library initialized successfully.
- 1: VXI library already initialized (repeat call).
- 1: Error occurred in InitVXIlibrary.

SetDevInfoLong

Sets information about a specified device in the device information table. The value being set is a 32-bit unsigned integer.



I16 la is the logical address of the device to set information for.

U16 field is the field identification number.

- 12: Base of A24/A32 memory.
- 13: Size of A24/A32 memory.

U32 longvalue is the information for that field.

I16**status.**

- 0: The specified information was set.
 - 1: Device not found.
 - 2: Invalid **field** specified.
-

SetDevInfoShort

Sets information about a specified device in the device information table. The value being set is a 16-bit unsigned integer.

**I16**

la is the logical address of the device to set information for.

U16

field is the field identification number.

- 2: Logical address of the Commander.
- 3: Mainframe.
- 4: Slot.
- 5: Manufacturer identification number.
- 7: Model code.
- 9: Device class.
- 10: Extended subclass (if extended class device).
- 11: Address space used.
- 14: Memory type and access time.
- 15: Bit vector list of VXI interrupter lines.
- 16: Bit vector list of VXI interrupt handler lines.
- 17: Mainframe extender, controller information.

Bits	Description
15 to 13	Reserved
12	1: Child side extender 0: Parent side extender
11	1: Frame extender 0: Not frame extender
10	1: Extended controller
9	1: Embedded controller
15 to 13	Reserved
12	1: Child side extender 0: Parent side extender

- 18: Asynchronous mode control state.
- 19: Response enable state.
- 20: Protocols supported.
- 21: Capability/status flags.
- 22: Status state (Pass/Fail, Ready/Not Ready).



shortvalue is the information for that field.



status.

- 0: The specified information was set.
- 1: Device not found.
- 2: Invalid **field** specified.

SetDevInfoStr

Sets information about a specified device in the device information table. The information being set is contained in a string.



I16

la is the logical address of the device to set information for.

U16

field is the field identification number.

- 1: Device name.
- 6: Manufacturer name.
- 8: Model name.

abc

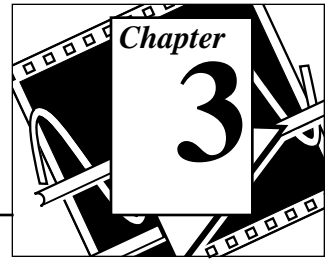
stringvalue is the buffer to receive information for that field.

I16

status.

- 0: The specified information was set.
- 1: Device not found.
- 2: Invalid **field** specified.

Word Serial Commander Protocol VIs



This chapter describes the VXI Word Serial Commander Protocol VIs. Word Serial communication is the minimal mode of communication between VXI Message-Based devices within the VXI Commander/Servant hierarchy.

Locating Word Serial Commander VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»Word Serial Commander Protocol** to locate the Word Serial Commander Protocol VIs in LabVIEW.

Finding Help Online for Word Serial Commander VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Word Serial Commander VI Descriptions

WSabort

Performs a Forced or Unrecognized (Unsupported) Command abort of a Word Serial operation(s) in progress.



I16

la is the logical address of the Message-Based device.

U16

abortop is the operation to abort.

- 1: Forced Abort: aborts WSwrt, WSrd, and WStrg.
- 2: UnSupCom: aborts WScmd, WSLcmd, and WSEcmd.
- 3: Forced Abort: aborts WScmd, WSLcmd, and WSEcmd.
- 4: Forced Abort: aborts all Word Serial operations.
- 5: Async Abort: aborts all Word Serial operations immediately.
Be careful when using this option. During a Word Serial query, the Servant may be left in an invalid state if the operation is aborted after writing the query and before reading the response register. When using this option, the Word Serial operation is aborted immediately as compared to using options 1, 3, and 4, where the operation is not aborted until the response is read in that situation.

I16

status.

- 0: Word Serial operation successfully aborted.
 - 1: Invalid **la**.
 - 2: Invalid **abortop**.
-

WSclr

Sends the Word Serial *Clear* command to a Message-Based device.



I16

la is the logical address of the Message-Based device.

[TF]

status is the return status bit vector (array of Boolean). The following table gives the meaning of each bit in the status array.

Index	Name	Description
7	BERR	Bus error occurred during transfer.
5	InvalidLA	Invalid logical address specified.
2	TIMO_DONE	Timed out before WR set (clear complete).
1	TIMO_SEND	Timed out before able to send command.
0	IODONE	Command transfer successfully completed.

WScmd

Sends a Word Serial command or query to a Message-Based device.



I16

la is the logical address of the Message-Based device.

U16

cmd is the Word Serial command value.



respflag is a Boolean value.

True: Get a response (query).

False: Do not get a response.



response is the 16-bit response.



status is the return status bit vector (array of Boolean). The following table gives the meaning of each bit in the status array.

Index	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	RRviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
7	BERR	Bus error occurred during transfer.
6	MQE	Multiple query error occurred during transfer
5	InvalidLA	Invalid logical address specified
2	TIMO_RES	Timed out before response received.
1	TIMO_SEND	Timed out before able to send command
0	IODONE	Command transfer successfully completed.

WSEcmd

Sends an Extended Longword Serial command or query to a Message-Based device.



I16

la is the logical address of the Message-Based device.

U16

cmdExt is the upper 16 bits of the 48-bit Extended Longword Serial command value.

U32

cmd is the lower 32 bits of the 48-bit Extended Longword Serial command value.

TF

respflag is a Boolean value.

True: Get a response (query).

False: Do not get a response.

U32

response is the 32-bit location to store the response.

[TF]

status is the return status bit vector (array of Boolean).

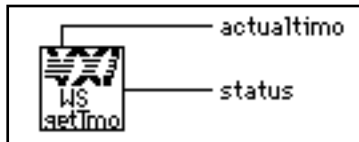
The following table gives the meaning of each bit in the status value.

Index	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	RRviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
7	BERR	Bus error occurred during transfer.

Index	Name	Description
6	MQE	Multiple query error occurred during transfer.
5	InvalidLA	Invalid logical address specified.
2	TIMO_RES	Timed out before response received.
1	TIMO_SEND	Timed out before able to send command.
0	IODONE	Command transfer successfully completed.

WSgetTmo

Gets the actual time period to wait before aborting a Word Serial, Longword Serial, or Extended Longword Serial Protocol transfer.



I32

actualtmo is the timeout period (in milliseconds).

I16

status.

0: Timeout period successfully received.

WSLcmd

Sends a Longword Serial command or query to a Message-Based device.



I16

la is the logical address of the Message-Based device.

U32

cmd is the Longword Serial command value.

TF

respflag is a Boolean value.

True: Get a response (query).
False: Do not get a response.

U32

response is the 32-bit location to store the response.

[TF]

status is the return status bit vector (array of Boolean).

The following table gives the meaning of each bit in the status value.

Index	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	RRviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
9	UnSupCom	Device did not recognize the command.
7	BERR	Bus error occurred during transfer.

Index	Name	Description
6	MQE	Multiple query error occurred during transfer.
5	InvalidLA	Invalid logical address specified.
2	TIMO_RES	Timed out before response received.
1	TIMO_SEND	Timed out before able to send command.
0	IODONE	Command transfer successfully completed.

WSLresp

Retrieves a response to a previously sent Longword Serial Protocol query from a VXI Message-Based device.

WSLcmd can send a query and automatically read a response. However, if you must break up the sending of the query and the reading of the response, you can use WSLcmd to send the query without reading the response and WSLresp to read the response.



Note: *This VI is intended for debugging purposes only.*



la is the logical address of the Message-Based device.



response is the 32-bit response.

[TF]

status is the return status bit vector (array of Boolean). The following table gives the meaning of each bit in the status value.

Index	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	RRviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
9	UnSupCom	Device did not recognize the command.
7	BERR	Bus error occurred during transfer.
6	MQE	Multiple query error occurred during transfer.
5	InvalidLA	Invalid logical address specified.
2	TIMO_RES	Timed out before response received.
0	IODONE	Command transfer successfully completed.

WSrd

Transfers the specified number of data bytes from a Message-Based device into a specified local memory buffer, using the VXIbus Byte Transfer Protocol.



I16

la is the logical address of the Message-Based device from which the buffer is read.

U32

count is the maximum number of bytes to transfer.

U16

mode is the transfer mode bit vector. The following table describes the mode bit vector corresponding to bits 15 through 0.

Bit	Event Signal
15 to 8	EOS character (valid if EOS termination)
4	EOS character termination 1: Terminate transfer on EOS bit. 0: Do not terminate transfer on EOS bit.
3	CR character termination 1: Terminate transfer on CR bit. 0: Do not terminate transfer on CR bit
2	LF character termination 1: Terminate transfer on LF bit. 0: Do not terminate transfer on LF bit
1	END bit termination suppression 0: Terminate transfer on END bit. 1: Do not terminate transfer on END bit.
0	Not DOR 0: Abort if not DOR. 1: Poll until DOR

abc

buf is the data read.

U32

retcount is the number of bytes actually transferred.

[TF]

status is the return status bit vector (array of Boolean). The following table gives the meaning of each bit in the status value.

Index	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	DORviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
9	UnSupCom	Device did not recognize the command.
8	TIMO	Timeout.
7	BERR	Bus error occurred during transfer.
6	MQE	Multiple query error occurred during transfer.
5	InvalidLA	Invalid logical address specified.
4	ForcedAbort	User abort occurred during I/O.
3	DirDorAbort	Transfer aborted—device is not DOR.
2	TC	All bytes received.
1	END	Any one of the termination received.
0	IODONE	Command transfer successfully completed.

WSresp

Retrieves a response for a previously sent Word Serial Protocol query from a VXI Message-Based device.

WScmd can send a query and automatically read a response. However, if it is necessary to break up the sending of the query and the reading of the response, you can use WScmd to send the query without reading the response and use WSresp to read the response.



Note: *This VI is intended for debugging purposes only.*



la is the logical address of the Message-Based device.



response is the 16-bit response.



status is the return status bit vector (array of Boolean).

The following table gives the meaning of each bit in the status value.

Item	Name	Description
14	WRviol	Write Ready protocol violation during transfer.
13	RRviol	Read Ready protocol violation during transfer.
12	DORviol	Data Out Ready protocol violation.
11	DIRviol	Data In Ready protocol violation.
10	RdProtErr	Read protocol error.
9	UnSupCom	Device does not support the command.
7	BERR	Bus error occurred during transfer.
6	MQE	Multiple query error occurred during transfer.
5	InvalidLA	Invalid logical address specified.

Item	Name	Description
2	TIMO_RES	Timed out before response received.
0	IODONE	Command transfer successfully completed.

WSsetTmo

Sets the time period to wait before aborting a Word Serial, Longword Serial, or Extended Longword Serial Protocol transfer. It returns the actual timeout value set (the nearest timeout period possible greater than or equal to the timeout period specified).



I32

timeout is the timeout period (in milliseconds).

I32

actual tmo is the actual timeout period set (in milliseconds).

I16

status.

0: Timeout value successfully set.

WStrg

Sends the Word Serial *Trigger* command to a Message-Based device.



I16

la is the logical address of the Message-Based device.

[TF]

status is the return status bit vector (array of Boolean).

The following table gives the meaning of each bit in the status value.

Item	Name	Description
14	WRviol	Write Ready protocol violation during the command transfer.
13	RRviol	Read Ready protocol violation occurred during the command transfer.
12	DORviol	Data Out Ready protocol violation occurred during the command transfer.
11	DIRviol	Data In Ready protocol violation occurred during the command transfer
10	RdProtErr	Read protocol error.
9	UnSupCom	Device did not recognize the command.
7	BERR	Bus error occurred during the command transfer.
6	MQE	Multiple query error occurred during the command transfer.
5	InvalidLA	Invalid logical address specified.
4	ForcedAbort	User abort occurred during I/O.
1	TIMO_SEND	Timed out before command transfer completed.
0	IODONE	Command transfer successfully completed.

WSwrt

Transfers the specified number of data bytes from a memory buffer to a Message-Based device, using the VXIbus Byte Transfer Protocol.



I16

la is the logical address of the Message-Based device to which the buffer is written.

abc

buf is the write buffer.

U16

mode is the transfer mode bit vector

for Bit 0

- 1: Poll until device is DIR.
- 0: Abort if device is not DIR.

for Bit 1

- 1: Set the END bit on the last byte of the transfer.
- 0: Clear the END bit on the last byte of the transfer.

U32

retcount is the number of bytes actually transferred.

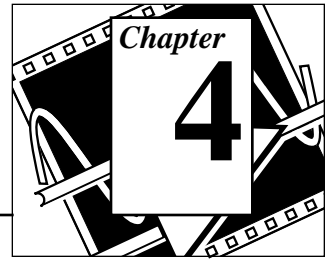
[TF]

status is the return status bit vector (array of Boolean). The following table gives the meaning of each bit in the status value.

Item	Name	Description
14	WRviol	Write Ready protocol violation occurred during the command transfer.
13	RRviol	Read Ready protocol violation occurred during the command transfer.
12	DORviol	Data Out Ready protocol violation occurred during the command transfer.

Item	Name	Description
11	DIRviol	Data In Ready protocol violation occurred during the command transfer.
10	RdProtErr	Read protocol error.
9	UnSupCom	Device did not recognize the command.
8	TIMO	Timeout.
7	BERR	Bus error occurred during the command transfer.
6	MQE	Multiple query error occurred during the command transfer.
5	InvalidLA	Invalid logical address specified.
4	ForcedAbort	User abort occurred during I/O.
3	DirDorAbort	Transfer aborted—Servant is not DIR.
2	TC	All bytes sent.
1	END	END sent with last byte.
0	IODONE	Transfer was successful.

Word Serial Servant Protocol VIs



This chapter describes the VXI Word Serial Servant Protocol VIs. Word Serial communication is the minimal mode of communication between VXI Message-Based devices within the VXI Commander/Servant hierarchy. The local CPU (the CPU on which the NI-VXI functions are running) uses the Word Serial Servant VIs to perform VXI Message-Based Word Serial Servant communication with its Commander.

You use these VIs only in the case where the local CPU is not a Top-Level Commander (probably not the Resource Manager), such as in a multiple CPU situation. In a multiple CPU situation, the local CPU must allow the Resource Manager device to configure the local CPU and can optionally implement some basic message-transfer Word Serial communication with its Commander. The four basic types of Word Serial Servant VIs are as follows:

- Command reception
- Query reception and responding
- Buffer sending
- Buffer receiving

Word Serial Protocol is a simple 16-bit transfer protocol between a Commander and its Servants. The Commander polls specific bits in the VXI Response register of the Servant to determine when a command can be written, when a response can be read from the Data Low register, and when a Word Serial protocol error occurs. Before a Commander can send a Word Serial command to a Servant, it must first poll the Write Ready (WR) bit until it is asserted (set to 1). The Commander can then write the command to the Data Low register.

If the Commander is sending a query, it first sends the query in the same manner as sending a command, but then continues by polling the Read Ready (RR) bit until it is asserted. It then reads the response from the Data Low register. A buffer write is simply a series of *Byte Available* Word Serial commands sent to the Servant, with the additional constraint that the Data In Ready (DIR) bit as well as the WR bit must be asserted before the *Byte Available* is sent. The lower 8 bits

(bits 0 to 7) of the 16-bit command contains a single byte of data (bit 8 is the END bit). Therefore, one *Byte Available* is sent for each data byte in the buffer written.

A buffer read is simply a series of *Byte Request* Word Serial queries sent to the Servant, with the additional constraint that the Data Out Ready (DOR) bit, as well as the WR bit, must be asserted before the *Byte Request* is sent. The lower 8 bits (bits 0 to 7) of the 16-bit response contain a single byte of data (bit 8 is the END bit). Therefore, one *Byte Request* is sent for each data byte in the buffer read.

In addition to the WR, RR, DIR, and DOR bits that get polled during various Word Serial transfers, the ERR* bit is also checked. The ERR* bit indicates that a Word Serial Protocol error has occurred. The Word Serial Protocol error can be Unsupported Command, Multiple Query Error (MQE), DIR Violation, DOR Violation, RR Violation, or WR Violation. The Word Serial Servant Protocol VIs allow the local CPU to generate any of the Word Serial Protocol errors and to respond to the *Read Protocol Error* Word Serial query with the corresponding protocol error. The ERR* bit assertion and unassertion are handled automatically.

The Longword Serial and Extended Longword Serial Protocols are similar to the Word Serial Protocol, but involve 32-bit and 48-bit command transfers, respectively, instead of the 16-bit transfers of the Word Serial Protocol. The VXI specification, however, provides no common command usages for these protocols. The commands are either VXI Reserved or User-Defined. The NI-VXI interface gives you the ability to receive and process any one of these commands.

Locating Word Serial Servant Protocol VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»Word Serial Servant Protocol** to locate the Word Serial Servant Protocol VIs in LabVIEW.

Finding Help Online for Word Serial Servant Protocol VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Word Serial Servant Protocol VI Descriptions

GenProtError

Generates a Word Serial protocol error if one is not already pending. The Response register bit ERR* is asserted if the value of the protocol error, **proterr**, is not FFFF. If **proterr** is FFFF, ERR* is unasserted. If no previous error existed, the **proterr** value is saved for response to a future *Read Protocol Error* query via the VI RespProtError.



proterr is the protocol error to generate an error.

- FFFF: Clear any protocol error condition.
- FFFD: Multiple Query Error (MQE).
- FFFC: Unsupported Command (UnSupCom).
- FFFB: Data In Ready violation (DIRviol).
- FFFA: Data Out Ready violation (DORviol).
- FFF9: Read Ready violation (RRviol).
- FFF8: Write Ready violation (WRviol).

Other values are reserved.

I16

status.

- 0: Successful.
 - 1: Previous error was pending, **proterr** ignored.
 - 1: Word Serial Servant VIs not supported.
-

GetWSScmdHandler

Returns the address of the current WSScmd interrupt handler.



U32

func is the address of the current WSScmd interrupt handler.

- 0: Word Serial Servant VIs not supported.
-

GetWSEcmdHandler

Returns the address of the current WSEcmd interrupt handler.



U32

func is the address of the current WSEcmd interrupt handler.

- 0: Word Serial Servant VIs not supported.
-

GetWSSLcmdHandler

Returns the address of the current WSSLcmd interrupt handler.



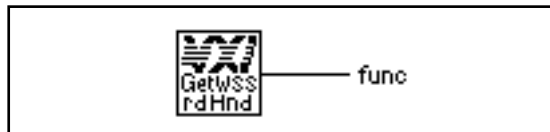
U32

func is the address of the current WSSLcmd interrupt handler.

0: Word Serial Servant VIs not supported.

GetWSSrdHandler

Returns the address of the current WSSrd done notification interrupt handler.



U32

func is the address of the current WSSrd done notification interrupt handler.

0: Word Serial Servant VIs not supported.

GetWSSwrtHandler

Returns the address of the current WSSwrt done notification interrupt handler.



U32

func is the address of the current WSSwrtdone notification interrupt handler.

0: Word Serial Servant VIs not supported.

PollWSScmdHandler

Waits until a Word Serial Protocol command or query is received from a VXI Message-Based Commander.



U16

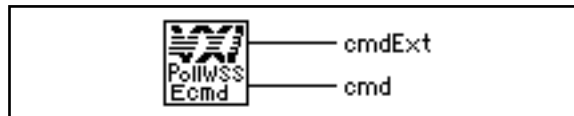
cmd is the 16-bit Word Serial command received.



Note: *If you need timeout capabilities with this VI, use the PollWSScmdHandlerTmo VI located in Chapter 13, VXIbus Supplemental VIs.*

PollWSSecmdHandler

Waits until an Extended Longword Serial Protocol command or query is received from a VXI Message-Based Commander.



U16

cmdExt is the upper 16 bits of the 48-bit Extended Longword Serial command value.

U32

cmd is the lower 32 bits of the 48-bit Extended Longword Serial command value.



Note: *If you need timeout capabilities with this VI, use the `PollWSSecmdHandlerTmo` VI located in Chapter 13, VXIbus Supplemental VIs.*

PollWSSLcmdHandler

Waits until a Longword Serial Protocol command or query is received from a VXI Message-Based Commander.



`cmd` is the 32-bit Longword Serial command received.



Note: *If you need timeout capabilities with this VI, use the `PollWSSLcmdHandlerTmo` VI located in Chapter 13, VXIbus Supplemental VIs.*

PollWSSrdHandler

Waits until a Word Serial Servant read operation (started with `WSSrd`) terminates.



`status`.

The following table gives the meaning of each bit in the status value if bit 15 is TRUE.

Bit	Error Conditions (Bit 15: 1)
14	Write Ready protocol violation during transfer
13	Read Ready protocol violation during transfer
12	Data Out Ready protocol violation
11	Data In Ready protocol violation
4	WSSabort called to force abort

The following table gives the meaning of each bit in the status value if bit 15 is FALSE.

Bit	Error Conditions (Bit 15: 0)
2	All bytes received
1	END received with last byte
0	Successful transfer



count is the actual number of bytes received.



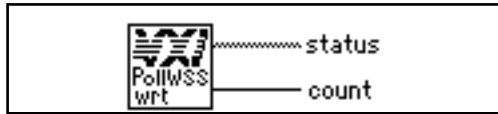
buf is the buffer received from WSSrd operation.



Note: *If you need timeout capabilities with this VI, use the `PollWSSrdHandlerTmo` VI located in Chapter 13, VXIbus Supplemental VIs.*

PollWSSwrthandler

Waits until a Word Serial Servant write operation (started with WSSwrthandler) terminates.



[TF] **status.**

The following table gives the value of each bit in the status value if bit 15 is TRUE.

Bit	Error Conditions (Bit 15: 1)
14	Write Ready protocol violation during transfer
13	Read Ready protocol violation during transfer
12	Data Out Ready protocol violation
11	Data In Ready protocol violation
4	WSSabort called to force abort

The following table gives the value of each bit in the status value if bit 15 is FALSE.

Bit	Error Conditions (Bit 15: 0)
2	All bytes sent
1	END sent with last byte
0	Successful transfer

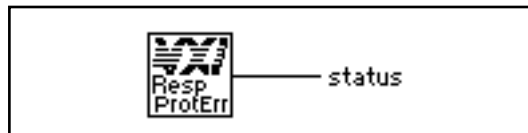
[U32] **count** is the actual number of bytes sent.



Note: *If you need timeout capabilities with this VI, use the **PollWSSwrHandlerTmo** VI located in Chapter 13, VXIbus Supplemental VIs.*

RespProtError

Responds to the Word Serial *Read Protocol Error* query with the last protocol error generated via the GenProtError VI. The ERR* bit is unasserted.



status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.
- 2: Response is still pending and a multiple query error is generated.

SetWSScmdHandler

Replaces the current WSScmd interrupt handler with a specified handler.



func is the address of the new WSScmd interrupt handler obtained from the GetWSScmdHandler VI.

- 0: Set to DefaultWSScmdHandler.
- 3: LabVIEW Occurrence Handler.

I16**status.**

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
-

SetWSSEcmdHandler

Replaces the current WSSEcmd interrupt handler with a specified handler.

**U32**

func is the address of the new WSSEcmd interrupt handler obtained from the GetWSSEcmdHandler VI.

- 0: Set to DefaultWSSEcmdHandler.
- 3: LabVIEW Occurrence Handler.

I16**status.**

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
-

SetWSSCmdHandler

Replaces the current WSSCmd interrupt handler with a specified handler.



U32

func is the address of the new WSSCmd interrupt handler obtained from the GetWSScmdHandler VI.

- 0: Set to DefaultWSSCmdHandler.
- 3: LabVIEW Occurrence Handler.

I16

status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.

SetWSSrdHandler

Replaces the current WSSrd done notification interrupt handler with a specified handler.



U32

func is the address of the new WSSrd done notification handler obtained from the GetWSScmdHandler VI.

- 0: Set to DefaultWSSrdHandler.
- 3: LabVIEW Occurrence Handler.

I16

status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.

SetWSSwrtHandler

Replaces the current WSSwrt done notification interrupt handler with a specified handler.



func is the address of the new WSSwrt done notification handler obtained from the GetWSScmdHandler VI.

- 0: Set to DefaultWSSwrtHandler.
- 3: LabVIEW Occurrence Handler.



status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.

WSSabort

Aborts the Word Serial Servant operation(s) in progress.



abortop is the operation to abort, bit vector.

Bit	Description
0	Abort WSSwrt
1	Abort WSSrd
2	Abort WSSsendResp

Bit	Description
15	Initialize Word Serial Servant hardware. This includes aborting all Word Serial operations, clearing out errors, removing all pending Word Serial Servant interrupts, and disabling the interrupts.

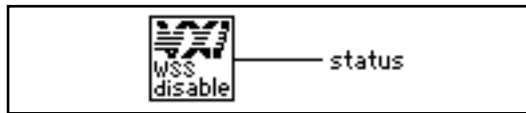


status.

- 0: Successfully aborted.
- 1: Word Serial Servant VIs not supported.
- 2: Unable to abort.

WSSdisable

Desensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.



status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.

WSSenable

Sensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.

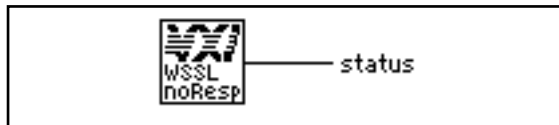


I16**status.**

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
-

WSSLnoResp

Acknowledges a received Longword Serial Protocol command that has no response and asserts the Write Ready (WR) bit in the local CPU Response register. You must call this VI after the processing of a Longword Serial Protocol command (queries are responded to with WSSLsendResp).

**I16****status.**

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
-

WSSLsendResp

Responds to a received Longword Serial Protocol query with a response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Longword Serial Protocol query (commands are acknowledged with WSSLnoResp).

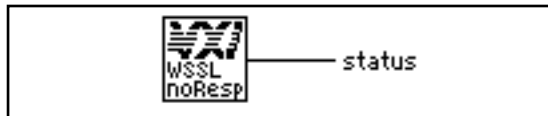
**U32****response** is a 32-bit response.

I16 status.

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
 - 2: Response still pending (MQE generated).
-

WSSnoResp

Acknowledges a received Word Serial Protocol command that has no response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Word Serial Protocol command (queries are responded to with WSSsendResp).



I16 status.

- 0: Successful.
 - 1: Word Serial Servant VIs not supported.
-

WSSrd

Posts a read operation to begin receiving the specified number of data bytes from a Message-Based Commander into a specified memory buffer, using the VXIbus Byte Transfer Protocol.



U32 **count** is the maximum number of bytes to transfer.

U16

mode is the transfer mode bit vector.

Bit	Description
0	Determines the DIR signal mode to Commander 0: Do not send DIR signal to Commander. 1: Send DIR signal to Commander.
1-15	0: Reserved.

I16

status.

- 0: Posted successfully.
- 1: Posted successfully, will not begin until WSSenable is called.
- 1: Word Serial Servant VIs not supported.
- 2: WSSrd already in progress.

WSSsendResp

Responds to a received Word Serial Protocol query with a response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Word Serial Protocol query (commands are acknowledged with WSSnoResp).

**U16**

response is the 16-bit response.

I16

status.

- 0: Successful.
- 1: Word Serial Servant VIs not supported.
- 2: Response still pending (MQE generated).

WSSwrt

Posts the write operation to transfer the specified number of data bytes from a specified memory buffer to the Message-Based Commander, using the VXIbus Byte Transfer Protocol.



U8

buf is the write buffer.

U16

mode is the mode of transfer (bit vector).

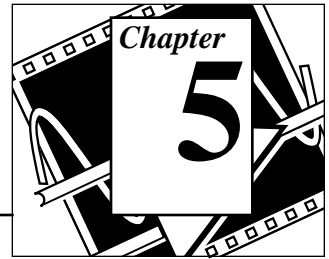
Bit	Description
0	Determines the DOR signal mode to Commander (if enabled). 0: Do not send DOR signal to Commander. 1: Send DOR signal to Commander.
1	Specifies the END bit termination with last byte. 0: Do not END with the last byte. 1: Send END with the last byte.
2-15	0: Reserved.

I16

status.

- 0: Posted successfully.
- 1: Posted successfully, will not begin until WSSenable is called.
- 1: Word Serial Servant VIs not supported.
- 2: WSSwrt already in progress.

Low-Level VXIbus Access VIs



This chapter describes how to use the VIs that give you the fastest access method for directly reading from or writing to any of the VXIbus address spaces.

There are several situations in which you must direct reads and writes to the different VXIbus address spaces, including some of the following.

- Register-Based device/instrument drivers
- Non-VXI/VME device/instrument drivers
- Accessing device-dependent registers on any type of VXI/VME device
- Implementing shared memory protocols

Low-level and high-level access to the VXIbus, as the NI-VXI interface defines them, are very similar in nature. Both sets of VIs can perform direct reads from and writes to any VXIbus address space with any privilege state or byte order. However, the two interfaces have different emphases with respect to user protection, error checking, and access speed.

Low-level VXIbus access is the fastest way to directly read from or write to the VXIbus address spaces. You access VXIbus address spaces by reading and writing to offsets in the local CPU address space that correspond to addresses on the VXIbus.

The address space of the local CPU is mapped onto the VXIbus in areas called *windows*. The size and number of windows varies, depending on the hardware. The size of the window is always a power of two, where a multiple of the size of the window would encompass an entire VXIbus address space. The *window base* register determines the multiple that a window can currently access.

The hardware platform determines the size of address space reserved for windows into VXIbus address spaces. Be sure to take into account the number and size of the windows provided by a particular platform. If mapping an address pointer requires the use of a window already

used by another address pointer, you must save and restore the window context. Before you increment or decrement a mapped pointer, test the bounds for accessing within a particular address space.



Note: *You should typically have the same access privileges and byte orders for all devices. The VXIbus specification requires that VXI devices respond to the supervisory data privilege state (address modifier codes). This increases the overall throughput of the program. Otherwise, the application must continually restore the state of the windows into VXIbus address spaces.*

NI-VXI uses a term within this chapter called the *hardware* (or *window*) *context*. The hardware context for a VXI window consists of the VXI address space being accessed, the base offset into the address space, the access privilege, and the byte order for the accesses through the window. Before accessing a particular address, you must set up the window with the appropriate hardware context using the MapVXIAddress VI. This VI returns an address pointer you can use for accessing the window in the future by using the VXIpeek and VXIpoke VIs.

Multiple Pointer Access for a Window

Sometimes problems occur when an application requires different privilege states, byte orders, and/or base addresses within the same window. If the hardware context is changed by a subsequent call to MapVXIAddress or by other VIs such as SetPrivilege or SetByteOrder, previously mapped windows would not have their intended access parameters. There are two types of access privileges to a window that help solve this problem: *Owner Privilege*, and *Access Only Privilege*. These two privileges define which caller of the MapVXIAddress VI can change the settings of the corresponding window.

Owner Privilege

A caller can obtain Owner Privilege to a window by requesting owner privilege in the MapVXIAddress VI (via the **accessparms** parameter). This address mapping will not succeed if another address pointer with Owner Privilege or Access Only Privilege has already been mapped for that window. If the mapping succeeds, the VI returns a valid pointer and a non-negative status value. The **window** output parameter returned from the MapVXIAddress VI associates the address pointer returned from the VI with a particular window and also signifies Owner

Privilege to that window. Owner Privilege access is complete and exclusive. The caller can use the SetPrivilege, SetByteOrder, and SetContext VIs with this window to dynamically change the access privileges. Notice that if the execution of the MapVXIAddress VI succeeds for either Owner Privilege or Access Only Privilege, the pointer remains valid in both cases until the UnMapVXIAddress VI is executed for the corresponding window. The advantage of Owner Privilege access is that it gives complete and exclusive access for that window to the caller, so you can dynamically change the access privileges. Because no other callers can succeed, there is no problem with destroying the access state of another caller.

Access Only Privilege

You can obtain Access Only Privilege for a window by requesting access only privileges in the MapVXIAddress VI. With this privilege mode, you can simultaneously have multiple address pointers to access a particular window, while still guaranteeing that the hardware context does not change between accesses. The VI executes successfully under either of the following conditions:

1. No address pointers are mapped for the window (first caller for Access Only Privilege for this window). The hardware context is set as requested in the call. The call returns a successful status and a valid address pointer and `window` for Access Only Privilege.
2. No address pointer has been mapped with Owner Privilege for the required window. There *are* address pointers with Access Only Privilege, but they are using the same hardware context (privilege state, byte order, address range) for their accesses to the window. Because the hardware context is compatible, it does not need to be changed. The VI returns a successful status and a valid address pointer and **window** for Access Only Privilege.

The successful call returns a valid pointer and a non-negative return value. The 32-bit window number signifies that the access privileges to the window are Access Only Privilege.

With Access Only Privilege, you cannot use the SetPrivilege, SetByteOrder, and SetContext VIs in your application to dynamically change the hardware context. No Access Only accessor can change the state of the window. The initial Access Only call sets the hardware context for the window, which cannot be changed until all Access Only accessors have called the UnMapVXIAddress VI to free the window.

The GetPrivilege, GetByteOrder, and GetContext VIs will succeed regardless of whether the caller has Owner Privilege or Access Only Privilege.

Locating Low-Level VXIbus Access VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»Low-Level VXIbus Access** to locate the Low-Level VXIbus Access VIs in LabVIEW.

Finding Help Online for Low-Level VXIbus Access VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Low-Level VXIbus Access VI Descriptions

GetByteOrder

Gets the byte/word order of data transferred into or out of the specified window.



window, as obtained from MapVXIAddress.



ordermode is the byte/word order of data.

- 0: Motorola byte ordering.
- 1: Intel byte ordering.

I16**status.**

- 0: Successful.
 - 1: Byte order returned successfully; same for all.
 - 1: Invalid **window**.
-

GetContext

Gets the current hardware interface settings (context) for the specified window.

**U32**

window, as obtained from MapVXIAddress.

U32

context is the VXI hardware access context.

I16**status.**

- 0: Context information was received successfully.
 - 1: Invalid window.
-

GetPrivilege

Gets the current VXI/VME access privilege for the specified window.

**U32**

window, as returned from MapVXIAddress.

U16 **priv** is the access privilege.

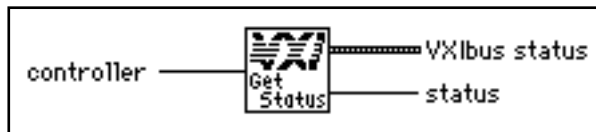
- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

I16 **status**.

- 0: Privilege access was received successfully.
- 1: Invalid **window**.

GetVXIbusStatus

Returns information about the state of the VXIbus in a specified controller (either an embedded CPU or an extended controller).



I16 **controller** is the controller from which to get the status (-2: OR of all).

ET **VXIbus status** is a cluster containing the VXIbus status.

The **VXIbus status** cluster consists of the following elements:

I16 **Bus Error**, where a value of 1 means that a bus error occurred on the last access.

I16 **Sysfail**, where a value of 1 means that SYSFAIL* is asserted.

I16 **ACfail**, where a value of 1 means that ACFAIL* is asserted.

I16 **Signal In** is the number of signals queued.



VXI ints is a bit vector, where a value of 1 in bit positions 0 through 6 means that the interrupt 1 through 7 is asserted.



ECL trigs is a bit vector, where a value of 1 in bit positions 0 through 5 means that the trigger 0 through 5 is asserted.



TTL trigs is a bit vector, where a value of 1 in bit positions 0 through 7 means that the trigger 0 through 7 is asserted.

A value of -1 returned in any of the fields of the cluster signifies that there is no hardware support to retrieve information for that particular VXIbus state.

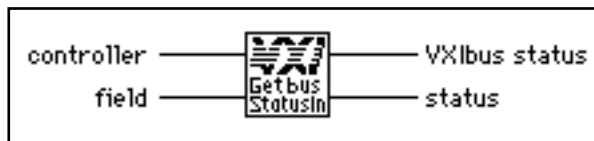


status.

- 0: Status information was received successfully.
- 1: Unsupported VI (no hardware support).
- 2: Invalid controller specified.

GetVXIbusStatusInd

Returns information about the state of the VXIbus for the specified field in a particular controller.



controller is the controller from which to get the status (-2: OR of all).



field is the number of the field about which to return information.

- 1: **Bus Error**, where a value of 1 means that the last access BERRed.
- 2: **Sysfail**, where a value of 1 means that SYSFAIL* is asserted.
- 3: **ACfail**, where a value of 1 means that ACFAIL* is asserted.
- 4: **Signal In**, which is the number of signals queued.
- 5: **VXI ints**, which is a bit vector, where a value of 1 in bit positions 0 through 6 means that the interrupt 1 through 7 is asserted.

- 6: **ECL trigs**, which is a bit vector, where a value of 1 in bit positions 0 through 5 means that the trigger 0 through 5 is asserted.
- 7: **TTL trigs** is a bit vector, where a value of 1 in bit positions 0 through 7 means that the trigger 0 through 7 is asserted.

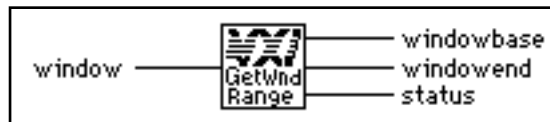
I16 **VXIbus status** in which a value of -1 in any of the fields means that there is no hardware support for that particular state.

I16 **status.**

- 0: Status information was received successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **controller** specified.
- 3: Invalid **field**.

GetWindowRange

Gets the range of addresses that a particular window, allocated with the MapVXIAddress VI, can currently access within a particular VXIbus address space.



U32 **window**, as obtained from MapVXIAddress.

U32 **windowbase** is the base VXI address.

U32 **windowend** is the end VXI address.

I16 **status.**

- 0: Window range successfully obtained.
- 1: Invalid **window**.

MapVXIAddress

Sets up a window into one of the VXI address spaces according to the access parameters specified, and returns a pointer to a local CPU address that accesses the specified VXI address. This VI also returns the window ID associated with the window, which is used with all other low-level VXIbus access VIs.



U16

accessparms specifies the access parameters.

Bits 0 and 1 are used to specify the VXI address space.

- 1: A16.
- 2: A24.
- 3: A32.

Bits 2 through 4 are used to specify the access privilege.

- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

Bit 5 = 0.

Bit 6 is used to specify the access mode.

- 0: Access only.
- 1: Owner access.

Bit 7 is used to specify the byte order.

- 0: Motorola.
- 1: Intel.

Bits 8 through 15 = 0.

U32 **address** is the address within A16, A24, or A32.

I32 **timeout** is the timeout (in milliseconds).

U32 **window** is the window number for use with other VIs.

I16 **status** displays the return status.

- 0: Map successfully created.
- 2: Invalid/unsupported **accessparms**.
- 3: Invalid **address**.
- 5: Byte order not supported.
- 6: Offset not accessible with this hardware.
- 7: Privilege not supported.
- 8: Timeout (**window** still in use; must use UnMapVXIAddress).

U32 **addressptr** is the pointer to local address for specified VXI address.

- 0: Unable to get pointer.



Note: *To maintain compatibility and portability, you should use the pointer returned by this VI only with the VXIpeek and VXIpoke VIs.*

SetByteOrder

Sets the byte/word order of data transferred into or out of the specified window.



U32 **window**, as obtained from MapVXIAddress.

U16 **ordermode** is the byte/word order of data.

- 0: Motorola byte ordering.
- 1: Intel byte ordering.

I16**status.**

- 0: Successful; byte order set for specific window only.
 - 1: Successful; byte order set for all windows.
 - 1: Invalid **window**.
 - 2: Invalid **ordermode**.
 - 5: **ordermode** not supported.
 - 9: **window** is not Owner Access.
-

SetContext

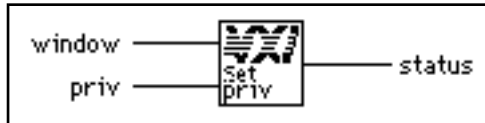
Sets the current hardware interface settings (context) for the specified window. The value for **context** should have been set previously by the GetContext VI.

**U32****window**, as returned from MapVXIAddress.**U32****context** is the VXI hardware access context to install, as returned from GetContext.**I16****status.**

- 0: Hardware interface settings set successfully.
 - 1: **window** does not exist.
 - 2: Invalid/unsupported **context**.
 - 9: **window** is not Owner Access.
-

SetPrivilege

Sets the VXI/VME access privilege for the specified window to the specified privilege state.



U32 **window**, as returned from MapVXIAddress.

U16 **priv** is the access privilege.

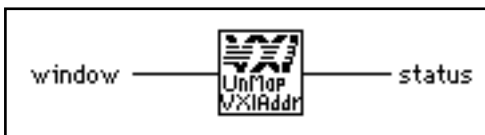
- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

I16 **status**.

- 0: Access privilege was set successfully.
- 1: Invalid window.
- 2: Invalid priv.
- 7: priv not supported.
- 9: window is not Owner Access.

UnMapVXIAddress

Deallocates a window that was allocated using the MapVXIAddress VI.



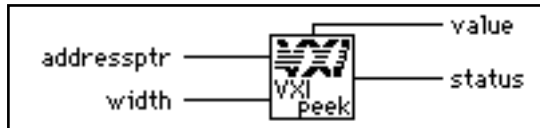
U32 **window**, as returned from MapVXIAddress.

I16**status.**

- 1: Access Only released (accessors remain).
 - 0: **window** successfully unmapped.
 - 1: Invalid **window**.
-

VXIpeek

Reads a single byte, word, or longword from a specified VXI address by de-referencing a pointer obtained from MapVXIAddress.

**U32**

addressptr is the address pointer obtained from MapVXIAddress.

U16

width specifies byte, word, or longword.

- 1: Byte.
- 2: Word.
- 4: Longword.

U32

value is the data value read.

I16**status.**

- 0: Transfer completed successfully.
 - 1: Bus error occurred during transfer.
-

VXIpoke

Writes a single byte, word, or longword to a specified VXI address by dereferencing a pointer obtained from MapVXIAddress.



U32

addressptr is the address pointer obtained from MapVXIAddress.

U16

width specifies byte, word, or longword.

- 1: Byte.
- 2: Word.
- 4: Longword.

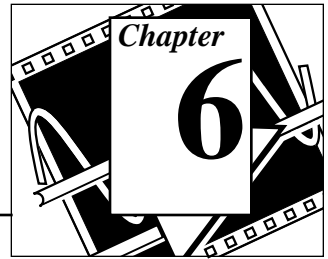
U32

value is the data value to write.

I16

status.

- 0: Transfer completed successfully.
- 1: Bus error occurred during transfer.



High-Level VXIbus Access VIs

This chapter describes the VIs with which you have direct access to the VXIbus address spaces. You can use these VIs to read, write, and move blocks of data between any of the VXIbus address spaces. Use these easy-to-use VIs when execution speed is not a critical issue.

Use low-level and high-level VXIbus Access VIs to directly read or write to VXIbus addresses. There are several situations in which you are required to direct reads and writes to the different VXIbus address spaces, including the following:

- Register-Based device/instrument drivers
- Non-VXI/VME device/instrument drivers
- Accessing device-dependent registers on any type of VXI/VME device
- Implementing shared memory protocols

Low-level and high-level access to the VXIbus, as the NI-VXI interface defines them, are very similar in nature. Both sets of VIs can perform direct reads of and writes to any VXIbus address space with any privilege state or byte order. However, the two interfaces have different emphases with respect to user protection, error checking, and access speed.

High-level VXIbus access VIs need not take into account any of the considerations that are required by the low-level VXIbus access VIs. The high-level VXIbus access VIs have all the necessary information for accessing a particular VXIbus address wholly contained within the VI input parameters. The parameters prescribe the address space, privilege state, byte order, and offset within the address space. Bus errors are automatically trapped, and an appropriate error status is returned.

More overhead is involved with the use of the high-level VXIbus access VIs, but if overall throughput of a particular access (for example, configuration or small number of accesses) is not the primary concern, the high-level VXIbus access VIs act as an easy-to-use interface that can do any VXIbus accesses necessary for an application.

All accesses to the VXIbus address spaces performed by use of the high-level VXIbus access VIs are fully protected. The hardware interface settings (*context*) for the applicable window are saved on entry to the VI and restored upon exit. No other VIs in the NI-VXI interface, including the low-level VXIbus access VIs, will conflict with the high-level VXIbus access VIs. You can use high-level and low-level VXIbus access VIs at the same time.

Locating High-Level VXIbus Access VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»High-Level VXIbus Access** to locate the High-Level VXIbus Access VIs in LabVIEW.

Finding Help Online for High-Level VXIbus Access VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

High-Level VXIbus Access VI Descriptions

VXlin

Reads a single byte, word, or longword from a specified VXI address with the specified byte order and privilege state.



U16**accessparms** specifies the access parameters.

Bits 0 and 1 are used to specify the VXI address space.

- 1: A16.
- 2: A24.
- 3: A32.

Bits 2 through 4 specify the access privilege.

- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

Bits 5 and 6 are reserved (should be 0).

Bit 7 specifies the byte order.

- 0: Motorola.
- 1: Intel.

Bits 8 through 15 are reserved (should be 0).

U32**address** is the VXI address within the specified space.**U16****width** is the read width.

- 1: Byte.
- 2: Word.
- 4: Longword.

U32**value** is the value read.**I16****status.**

- 0: Read completed successfully.
- 1: Bus error occurred during transfer.
- 2: Invalid **accessparms**.
- 3: Invalid **address**.
- 4: Invalid **width**.

- 5: Byte order not supported.
 - 6: **address** not accessible with this hardware.
 - 7: Privilege not supported.
 - 9: Width not supported.
-

VXlinReg

Reads a single word from a specified VXI register offset on the specified VXI device. The register is read in Motorola byte order as nonprivileged data.



la is the logical address of the device to read from.



reg is the offset within VXI logical address registers.



value is the value read from the VXI register of the device.

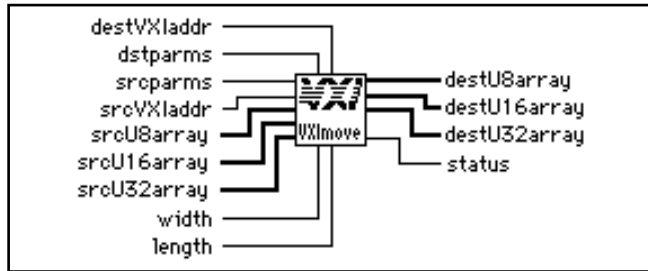


status.

- 0: Read completed successfully.
 - 1: Bus error occurred during transfer.
 - 3: Invalid **reg** specified.
-

VXImove

Copies a block of memory from a specified source location in any address space (local, A16, A24, A32) to a specified destination in any address space.



destVXIaddr is the destination address in the VXI address space (applicable only if the address space specified by **dstparms** indicates VXI address space).



dstparms specifies the destination parameters.

Bits 0 and 1 are used to specify the destination address space

- 0: Local (bits 2, 3, 4, and 7 should be 0).
- 1: A16.
- 2: A24.
- 3: A32.

Bits 2 through 4 are used to specify the access privilege.

- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

Bits 5 and 6 are reserved (should be 0).

Bit 7 is used to specify the byte order.

- 0: Motorola.
- 1: Intel.

Bits 8 through 15 are reserved (should be 0).



srcparms specifies the source parameters.

Bits 0 and 1 are used to specify the source address space.

- 0: Local (bits 2, 3, 4, and 7 should be 0).
- 1: A16.
- 2: A24.
- 3: A32.

Bits 2 through 4 are used to specify the access privilege.

- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

Bits 5 and 6 are reserved (should be 0).

Bit 7 is used to specify the byte order.

- 0: Motorola.
- 1: Intel.

Bits 8 through 15 are reserved (should be 0).



srcVXIaddr is the source address in the VXI address space (applicable only if the address space specified by **srcparms** indicates VXI address space).



srcU8array is the source unsigned byte array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of bytes).



srcU16array is the source unsigned word array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of words).



srcU32array is the source unsigned longword array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of longwords).

U16**width** specifies byte, word, or longword.

- 1: Byte.
- 2: Word.
- 4: Longword.

U32**length** is the number of elements to transfer.**U8**

destU8array is the destination unsigned byte array in the local address space (applicable only if the address space specified by **destparms** indicates local address space, and you want to transfer data into an array of bytes).

U16

destU16array is the destination unsigned word array in the local address space (applicable only if the address space specified by **destparms** indicates local address space, and you want to transfer data into an array of words).

U32

destU32array is the destination unsigned longword array in the local address space (applicable only if the address space specified by **destparms** indicates local address space and you want to transfer data into an array of longwords).

I16**status.**

- 0: Transfer completed successfully.
 - 1: Bus error occurred during transfer.
 - 2: Invalid **srcparms** or **destparms**.
 - 3: Invalid **srcVXIaddr** or **destVXIaddr**.
 - 4: Invalid **width**.
 - 5: Byte order not supported.
 - 6: Address not accessible with this hardware.
 - 7: Privilege not supported.
 - 8: Timeout; DMA aborted (if applicable).
 - 9: Width not supported.
-

VXlout

Writes a single byte, word, or longword to a specified VXI address with the specified byte order and privilege state.



U16

accessparms specifies the access parameters.

Bits 0 and 1 are used to specify the VXI address space.

- 1: A16.
- 2: A24.
- 3: A32.

Bits 2 through 4 are used to specify the access privilege.

- 0: Nonprivileged data access.
- 1: Supervisory data access.
- 2: Nonprivileged program access.
- 3: Supervisory program access.
- 4: Nonprivileged block access.
- 5: Supervisory block access.

Bits 5 and 6 are reserved (should be 0).

Bit 7 specifies the byte order.

- 0: Motorola.
- 1: Intel.

Bits 8 through 15 are reserved (should be 0).

U32

address is the VXI address within the specified space.

U16

width is the write width.

- 1: Byte.
- 2: Word.
- 4: Longword.

U32**value** is the data value to write.**I16****status.**

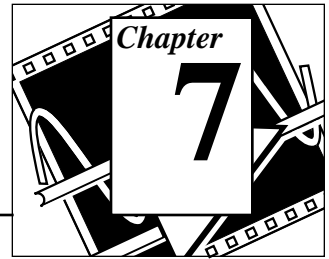
- 0: Write completed successfully.
 - 1: Bus error occurred during transfer.
 - 2: Invalid **accessparms**.
 - 3: Invalid **address**.
 - 4: Invalid **width**.
 - 5: Byte order not supported.
 - 6: Address not accessible with this hardware.
 - 7: Privilege not supported.
 - 9: Width not supported.
-

VXloutReg

Writes a single word to a specified VXI register offset on the specified VXI device. The register is written in Motorola byte order and as nonprivileged data.

**I16****la** is the logical address of the device to write to.**U16****reg** is the offset within VXI logical address registers.**U16****value** is the value written to the VXI register of the device.**I16****status.**

- 0: Write completed successfully.
 - 1: Bus error occurred during transfer.
 - 3: Invalid **reg** specified.
-



Local Resource Access VIs

This chapter describes the VIs you use to access miscellaneous local resources such as the local CPU VXI register set, Slot 0 MODID operations, and the local CPU VXI Shared RAM. These VIs are useful for shared memory type communication, non-Resource Manager operation, and debugging purposes.

Access to the local logical address of the CPU is required for sending correct VXI signal values to other devices. Reading local VXI registers is required for retrieving configuration information. Writing to the A24 and A32 pointer registers is required for use under the Shared Memory Protocol of the VXIbus specification, Revision 1.2. Exercising the local CPU MODID capabilities (if the local CPU is a VXI Slot 0 device) can be helpful in debugging the slot association (MODID) capability of a prototype VXI device.

Locating Local Resource Access VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»Local Resource Access** to locate the Local Resource Access VIs in LabVIEW.

Finding Help Online for Local Resource Access VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Local Resource Access VI Descriptions

GetMyLA

Gets the logical address of the local VXI device (the VXI device on which this copy of the NI-VXI software is running).

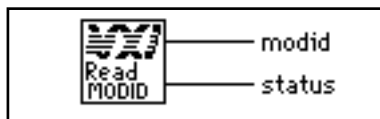


I16

la is the logical address of the local device.

ReadMODID

Senses the MODID lines of the VXIbus backplane. This VI applies only to the local device, which must be a Slot 0 device.



U16

modid is a bit vector.

Bits 12 through 0 correspond to MODID lines 12 through 0, respectively, and Bit 13 is the MODID enable bit.

I16

status.

- 0: MODID lines read successfully.
 - 1: Not a Slot 0 device.
-

SetMODID

Controls the assertion of the MODID lines of the VXIbus backplane. This VI applies only to the local device, which must be a Slot 0 device.



U16

enable defines the handling of the MODID enable bit.

- 1: Set MODID enable bit.
- 0: Clear MODID enable bit.

U16

modid is a bit vector for Bits 12 through 0, corresponding to Slots 12 through 0, respectively.

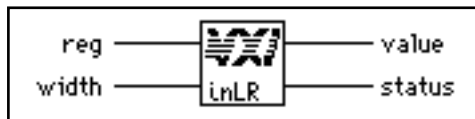
I16

status.

- 0: MODID lines set successfully.
- 1: Not a Slot 0 device.

VXlinLR

Reads a single byte, word, or longword from a particular VXI register on the local VXI device. The register is read in Motorola byte order and as nonprivileged data.



U16

reg is the offset within VXI logical address registers.

U16

width specifies byte, word, or longword.

- 1: Byte.
- 2: Word.
- 4: Longword.

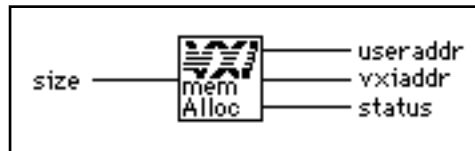
U32 **value** is the data value read.

I16 **status.**

- 0: Read completed successfully.
 - 1: Bus error.
 - 3: Invalid **reg**.
 - 4: Invalid **width**.
 - 9: **width** not supported.
-

VXImemAlloc

Allocates dynamic system RAM from the VXI Shared RAM area of the local CPU and returns both the local and remote VXI addresses. The VXI address space is the same as the space for which the local device is porting memory. You can use this VI to set up shared memory transfers.



U32 **size** is the number of bytes to allocate.

U32 **useraddr** is the returned application memory buffer address.

(This buffer cannot be directly accessed by LabVIEW. Use the VXImemCopy VI to access data in this buffer.)

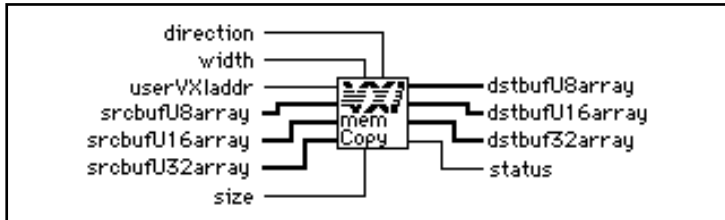
U32 **vxiaddr** is the returned remote VXI memory buffer address.

I16 **status.**

- 0: Successful.
 - 1: Successful; memory must be accessed using VXImemCopy.
 - 1: Memory allocation failed.
 - 2: Local CPU is A16 only.
-

VXImemCopy

Copies data into the local memory array from the VXI Shared RAM area of the local CPU allocated by VXImemAlloc or copies data from the local memory array into the VXI Shared RAM.



[U16]

direction designates the area from which you want to copy data.

- 1: Copy data from the specified source array to the **userVXIaddr** parameter.
- 0: Copy data from the **userVXIaddr** parameter into the destination array specified by the **width** parameter.

[U16]

width is the copy width.

- 1: Byte.
- 2: Word.
- 4: Longword.

[U32]

userVXIaddr is the Shared RAM address to copy data from or into.

[U8]

srcbufU8array is the source unsigned byte array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of bytes into the Shared RAM).

[U16]

srcbufU16array is the source unsigned word array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of words into the Shared RAM).

[U32]

srcbufU32array is the source unsigned longword array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of longwords into the Shared RAM).

[U32]

size is the number of elements to copy.

[U8]

dstbufU8array is the destination unsigned byte array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data into an array of bytes from the Shared RAM and the **width** parameter indicates byte transfers).

[U16]

dstbufU16array is the destination unsigned word array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data into an array of words from the Shared RAM and the **width** parameter indicates word transfers).

[U32]

dstbufU32array is the destination unsigned longword array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of longwords from the Shared RAM and the **width** parameter indicates longword transfers).

[I16]

status.

- 0: Successful.
- 1: Error in copying data.
- 2: Local CPU is A16 only.
- 5: Invalid **direction**.

VXImemFree

Deallocates dynamic system RAM from the VXI Shared RAM area of the local CPU that was allocated using the VXImemAlloc VI.

**[U32]**

useraddr is the application memory buffer address to free.

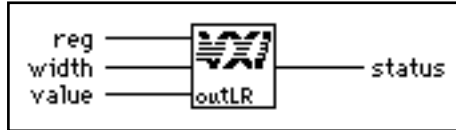
[I16]

status.

- 0: Successful.
- 1: Memory deallocation failed.

VXIoutLR

Writes a single byte, word, or longword to a particular VXI register on the local VXI device. The register is written in Motorola byte order and as nonprivileged data.



U16

reg is the offset within VXI logical address registers.

U16

width specifies byte, word, or longword.

- 1: Byte.
- 2: Word.
- 4: Longword.

U32

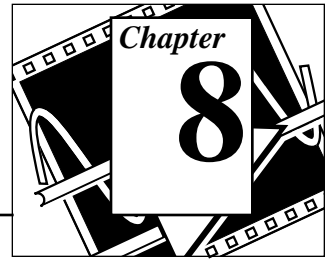
value is the data value to write.

I16

status.

- 0: Read completed successfully.
- 1: Bus error.
- 3: Invalid **reg**.
- 4: Invalid **width**.
- 9: **width** not supported.

VXI Signal VIs



This chapter describes the VIs you use to specify signal routing, manipulate the global signal queue, and wait for a particular VXI signal to be received.

VXI signals are a basic form of asynchronous communication that VXI bus master devices use. A VXI signal is simply a 16-bit value written to the Signal register of a VXI Message-Based device. Normally, the write to the Signal register generates a local CPU interrupt, and the local CPU then acquires the signal value in some device-specific manner. All National Instruments hardware platforms have a hardware FIFO to accumulate signal values while waiting for the local CPU to retrieve them. The format of the 16-bit signal value is defined by the VXIbus specification. VXI signals and status/ID values contain the VXI logical address of the sending device in the lower 8 bits of the VXI signal or status/ID value. The upper 8 bits of the 16-bit value depends on the VXI device type.

Three methods are available for handling VXI signals in LabVIEW. One method is to allow the NI-VXI default signal handler to handle the signal. From the point of view of your LabVIEW application, using the default signal handler essentially results in the signal being ignored.

A second method for handling signals is by queuing on a global signal queue. The RouteSignal VI specifies which types of signals are handled by the default signal handler and which signals are placed on the global signal queue. (By default, when signals are initially enabled with the EnableSignalInt VI, all signals are routed to the default signal handler.) The VIs used to access the signal queue are SignalDeq, SignalEnq, and SignalJam.

The third method for handling signals is with the WaitForSignal VI. This VI can be used to suspend the execution of a VI until a particular signal (or one of a set of signals) arrives. In LabVIEW, any number of WaitForSignal VIs can be executed in parallel, even for the same logical address. When using the WaitForSignal VI, you should use RouteSignal to route the desired signals to the global signal queue. The

WaitForSignal VI will first check the queue to see if the signal(s) in which you are interested have already been received.

Locating VXI Signal VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»VXI Signal** to locate the VXI Signal VIs in LabVIEW.

Finding Help Online for VXI Signal VIs

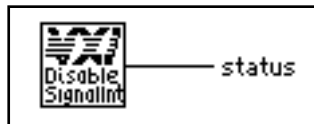
You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

VXI Signal VI Descriptions

DisableSignalInt

Desensitizes the local CPU to interrupts generated by writes to the local VXI Signal register. While disabled, no VXI signals are processed. If the local VXI hardware Signal register is implemented as a FIFO, signals are held in the FIFO until the signal interrupt is enabled via the EnableSignalInt VI.



I16**status.**

0: Signal interrupts were successfully disabled.

EnableSignalInt

Sensitizes the local CPU to interrupts generated by writes to the local VXI Signal register.

**I16****status.**

1: Signal queue is full; will enable after a SignalDeq.
 0: Signal interrupts were successfully enabled.

GetSignalHandler

Returns the address of the current signal interrupt handler for a specified logical address.

**U16**

la signifies the logical address for finding the address of the signal interrupt handler.

-2: Unknown (miscellaneous) signal source.

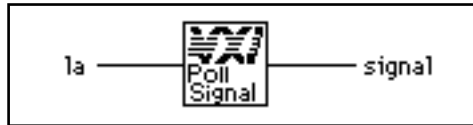
U32

func is the pointer to the current signal interrupt handler for the specified logical address.

0: invalid **la**.

PollSignalHandler

Waits until a VXI signal from the specified logical address (**la**) is received and then continues.



la specifies the logical address of the signals to be handled.

- 1: All known logical addresses.
- 2: Unknown (miscellaneous) signal sources.



signal specifies the actual 16-bit VXI signal.



Note: *If you need timeout capabilities with this VI, use the **PollSignalHandlerTmo** VI located in Chapter 13, VXIbus Supplemental VIs.*

RouteSignal

Specifies how each type of signal is to be processed for each logical address. A signal can be enqueued on a global signal queue (for later dequeuing via SignalDeq) or handled at interrupt service routine time by the default signal handler.



la specifies the routing for the logical address.

- 1: All known logical addresses.



modemask is a bit vector that specifies whether each type of signal is enqueued or handled by the signal handler.

A zero in any bit position causes signals of the associated type to be queued on the global signal queue, and all other signals are handled by the signal handler.

The following table describes the event signals that correspond to bits 14 through 8 when **la** is a Message-Based device.

Bit	Event Signal
14	User-defined Events
13	VXI Reserved Events
12	Shared Memory Events
11	Unrecognized Command Events
10	Request False (REQF) Events
9	Request True (REQT) Events
8	No Cause Given Events.

The following table shows the response signals that correspond to bits 7 through 0 when **la** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

The following table shows the type of signal values that correspond to bits 15 through 0 when **la** is a *non-Message-Based* device.

Bit	Type of Signal (status/ID) Values
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

I16**status.**

- 0: Signal routed successfully.
- 1: Invalid **la**.

SetSignalHandler

Replaces the current signal interrupt handler for a logical address with a specified handler.

**I16****la** specifies the logical address to set the handler to.

- 1: All known logical addresses.
- 2: Unknown (miscellaneous) signal handler.

U32**func** is the pointer to the new signal interrupt handler.

- 0: Set to DefaultSignalHandler.
- 3: LabVIEW occurrence handler.

I16**status.**

- 0: Successful.
- 1: Invalid **la**.

SignalDeq

Gets a signal specified by the signalmask from the signal queue for the specified logical address.



I16

la specifies the logical address from which to dequeue signal.

- 255: VME interrupt routed to signal queue.
- 1: any known **la**.

U32

signalmask is a bit vector that specifies the types of signals to dequeue.

A one in any bit position causes the subroutine to dequeue signals of the associated type.

The following table describes the event signals that correspond to bits 14 through 8 when **la** is a Message-Based device.

Bit	Event Signal
14	User-defined Events
13	VXI Reserved Events
12	Shared Memory Events
11	Unrecognized Command Events
10	Request False (REQF) Events
9	Request True (REQT) Events
8	No Cause Given Events

The following table shows the response signals that correspond to bits 7 through 0 when **ia** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

The following table shows the type of signal values that correspond to bits 15 through 0 when **ia** is a *non*-Message-Based device, or if **ia** = 255 (VME status/ID).

Bit	Event Signal
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)

U16

signal is the signal value dequeued from the signal queue.

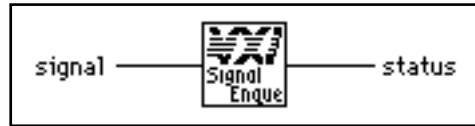
I16

status.

- 0: A signal was returned in **signal**.
- 1: The signal queue is empty or no match.

SignalEnq

Puts a signal on the tail of the signal queue for the specified logical address.



U16

signal is the value to enqueue at the tail of the signal queue.

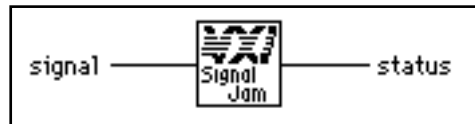
I16

status.

- 0: The signal was queued.
 - 1: The signal was not queued because the signal queue is full.
 - 2: The signal was not queued because the logical address is invalid.
-

SignalJam

Puts a signal on the head of the signal queue for the specified logical address.



Note: *This VI is intended for debugging purposes only.*

U16

signal is the signal value to put on the head of the queue.

I16

status.

- 0: The signal was queued.
 - 1: The signal was not queued because the signal queue is full.
 - 2: The signal was not queued because the logical address is invalid.
-

WaitForSignal

Waits for a specified type of signal to be received from a specified logical address. A signal mask defines the type(s) of signals that the application program waits for. The timeout value specifies the maximum amount of time (in milliseconds) to wait until the signal occurs.



I16

la is the logical address of the device sourcing the signal.

- 255: VME interrupt routed to signal queue.
- 1: any known **la**.

U32

signalmask is a bit vector that indicates the type(s) of signals that the application will wait for.

A one in any bit position causes the subroutine to detect signals of the associated type.

The following table shows the event signals that correspond to bits 14 through 8, when **la** is a Message-Based device.

Bit	Event Signal
14	User-defined Events
13	VXI Reserved Events
12	Shared Memory Events
11	Unrecognized Command Events
10	Request False (REQF) Events
9	Request True (REQT) Events
8	No Cause Given Events

The following table shows the response signals that correspond to bits 7 through 0, when **la** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (Reserved for future definition)
5	Data Out Ready (DOR)
4	Data In Ready (DIR)
3	Protocol Error (ERR)
2	Read Ready (RR)
1	Write Ready (WR)
0	Fast Handshake (FHS)

The following table shows the type of signal values that correspond to bits 15 through 0 when **la** is a *non*-Message-Based device, or if **la** = 255 (VME status/ID).

Bit	Event Signal
15 to 8	Active high bit (if 1 in bits 15 to 8, respectively)
7 to 0	Active low bit (if 0 in bits 15 to 8, respectively)



timeout is the maximum amount of time (in milliseconds) to wait until the signal occurs.

0: Forever.



retsignal is the signal received (upper byte of the 16-bit signal).



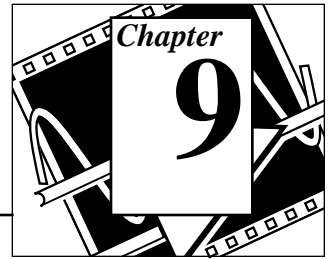
retsignalmask is a bit vector that indicates the type(s) of signals that the application received. The bits have the same meanings as given for the input parameter **signalmask**.



status.

- 0: One of the specified signals was received.
 - 1: Invalid **ia**.
 - 2: Timeout occurred while waiting for the specified signal(s).
-

VXI Interrupt VIs



This chapter describes the VIs that control VXI interrupts. VXI interrupts are a basic form of asynchronous communication used by VXI devices with VXI interrupter support. In VME, a device asserts a VME interrupt line, and the VME interrupt handler device acknowledges the interrupt. During the VME interrupt acknowledge cycle, a status/ID value is returned.

On most 680X0-based VME CPUs, this value is used as a local interrupt vector value and routed directly to the 680X0 processor. This value is used to look up which interrupt service routine to invoke. In VXI, however, the VXI interrupt acknowledge cycle returns (at a minimum) a 16-bit status/ID value. This 16-bit status/ID value is data, not a vector base location. The definition of the 16-bit vector is specified by the VXIbus specification and is the same as for a VXI signal. The lower 8 bits of the status/ID value is the VXI logical address of the interrupting device, while the upper 8 bits specifies the reason for interrupting.

Because the interrupt status/ID value for a VXI device is the same as a VXI signal value, VXI interrupts can be handled as VXI signals. The `RouteVXIint` VI is used to specify that a VXI interrupt be handled as a VXI signal. If a VXI interrupt is not routed to be processed as a VXI signal, the interrupt will be handled by the NI-VXI default VXI interrupt handler, resulting in your LabVIEW application essentially ignoring the interrupt.

To sensitize and desensitize the LabVIEW application to VXI interrupts routed to the VXI interrupt handlers, use the `EnableVXIint` and `DisableVXIint` VIs, respectively. To enable queuing of the interrupt after executing `RouteVXIint`, you must execute the `EnableVXItoSignalInt` VI. Once the VXI interrupt is routed and enabled, the interrupt is handled as a VXI signal using the signal queue VIs. To disable queuing of the interrupt, you must execute the `DisableVXItoSignalInt` VI. For more information concerning VXI signal VIs, refer to Chapter 8, *VXI Signal VIs*.

Locating VXI Interrupt VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»VXI Interrupt** to locate the VXI Interrupt VIs in LabVIEW.

Finding Help Online for VXI Interrupt VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

VXI Interrupt VI Descriptions

AcknowledgeVXlint

Performs an IACK cycle on the VXIbus on the specified controller (either an embedded CPU or an extended controller) for a particular VXI interrupt level.



VXI interrupts are automatically acknowledged when enabled by the EnableVXItoSignalInt and EnableVXIint VIs. Use this VI to manually acknowledge VXI interrupts that the local device is not enabled to receive.



Note: *This VI is intended for debugging purposes only.*



controller is the controller on which to acknowledge the interrupt.



level is the interrupt level to acknowledge.

U32**statusID** is the status/ID obtained during the IACK cycle.**I16****status.**

- 0: The IACK cycle completed successfully.
 - 1: Unsupportable VI (no hardware support for IACK).
 - 2: Invalid **controller**.
 - 3: Invalid **level**.
 - 4: Bus error occurred during the IACK cycle.
-

AssertVXIint

Asserts a VXI interrupt line on the specified controller (either an embedded CPU or an extended controller).



When the VXI interrupt is acknowledged (a VXI IACK cycle occurs), the specified status/ID is passed to the device that acknowledges the VXI interrupt.

I16**controller** is the controller on which to assert the interrupt.**U16****level** is the interrupt level to assert.**U32****statusID** is the status/ID to present during the IACK cycle.**I16****status.**

- 0: The interrupt line was asserted successfully.
 - 1: Unsupportable VI (no hardware support for VXI interrupter).
 - 2: Invalid **controller**.
 - 3: Invalid **level**.
 - 5: VXI interrupt still pending from the previous AssertVXIint VI.
-

DeAssertVXlint

Asynchronously deasserts a VXI interrupt line on the specified controller (either an embedded CPU or an extended controller) that was previously asserted by the AssertVXInt VI.



Note: *This VI is intended for debugging purposes only. Deasserting a VXI interrupt can cause a violation of the VME and VXIbus specifications.*



controller is the controller on which to deassert the interrupt.



level is the interrupt level to deassert.



status.

- 0: The interrupt line was deasserted successfully.
- 1: Unsupportable VI (no hardware support for VXI interrupter).
- 2: Invalid **controller**.
- 3: Invalid **level**.

DisableVXInt

Desensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI interrupts (not as signals) via the RouteVXInt VI.



controller specifies the controller (embedded or extended) to disable the interrupts.



levels is a vector of VXI interrupt levels to disable.

- 1: Disable for appropriate level.

0: Leave at current setting.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

I16

status.

- 0: VXI interrupt successfully disabled.
- 1: No hardware support.
- 2: Invalid **controller**.

DisableVXItoSignalInt

Desensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI signals (not as interrupts) via the RouteVXIint VI.



I16

controller specifies the controller (embedded or extended) to disable the interrupts.

U16

levels is a vector of VXI interrupt levels to disable.

- 1: Disable for appropriate level.
- 0: Leave at current setting.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

I16

status.

- 0: VXI interrupt successfully disabled.
- 1: No hardware support.
- 2: Invalid **controller**.

EnableVXIint

Sensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI interrupts (not as signals) via the RouteVXIint VI.



The RM assigns the interrupt levels automatically. Use the GetDevInfoShort VI to retrieve the assigned levels. Notice that each VXI interrupt is physically enabled only if the RouteVXIint VI has specified that the VXI interrupt be routed and then handled as a VXI/VME interrupt.



controller specifies the controller (embedded or extended) to enable the interrupts.



levels is a vector of VXI interrupt levels to enable.

- 1: Enable for appropriate level.
- 0: Leave at current setting.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.



status.

- 0: VXI interrupt successfully enabled.
- 1: No hardware support.
- 2: Invalid **controller**.

EnableVXItoSignalInt

Sensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI signals (not as interrupts) via the RouteVXIint VI.



The RM assigns the interrupt levels automatically. GetDevInfoShort can be used to retrieve the assigned levels. Notice that each VXI interrupt is physically enabled only if the RouteVXIint VI has specified that the VXI interrupt be routed and then handled as a VXI/VME signal.



controller specifies the controller (embedded or extended) to enable the interrupts.



levels is a vector of VXI interrupt levels to enable.

- 1: Enable for appropriate level.
- 0: Leave at current setting.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.



status.

- 1: Signal queue is full, will enable after a SignalDeq.
- 0: VXI interrupt successfully enabled.
- 1: No hardware support.
- 2: Invalid **controller**.

GetVXIintHandler

Returns the address of the current interrupt handler for a specified VXIbus interrupt level.



level is the VXI interrupt level associated with the handler.

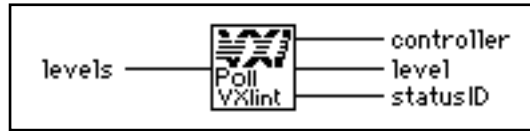


func is the pointer to the current interrupt handler for a specified VXIbus interrupt level.

- 0: Invalid **level** or no hardware support.

PollVXIintHandler

Waits until a VXI interrupt occurs and then continues.



U16

levels is a bit vector that specifies which levels to poll.

- 1: set.
- 0: do not set.

Bits 0 to 6 correspond to VXI interrupt levels 1 to 7 respectively.

I16

controller specifies the logical address of controller interrupting.

U16

level specifies the received VXI interrupt level.

U32

statusID specifies the status/ID obtained during IACK cycle (if it is a 16-bit VXI IACK value, it may be equivalent to a VXI signal).



Note: *If you need timeout capabilities with this VI, use the **PollVXIintHandlerTmo** VI located in Chapter 13, VXIbus Supplemental VIs.*

RouteVXIint

Specifies whether the status/ID value retrieved from a VXI interrupt acknowledge cycle is routed to the VXI interrupt handler or to the signal processing routine.



The RouteVXIint VI dynamically enables and disables the appropriate VXI interrupts based on the settings from calls to the EnableVXItoSignalInt and EnableVXIint VIs.



controller specifies the controller (embedded or extended) to enable the interrupts.



Sroute is a bit vector that specifies whether to handle a VXI interrupt as a signal or route it to the VXI interrupt handler routine.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

- 1: Handle VXI interrupt for this level as a signal.
- 0: Handle VXI interrupt as a VXI interrupt.

Bits 14 to 8 correspond to VXI interrupt levels 7 to 1, respectively.

- 1: Route as 8-bit VME status/ID.
- 0: Route as 16-bit VXI status/ID.



status.

- 0: Status/ID value routed successfully.
- 1: No hardware support.
- 2: Invalid **controller**.

SetVXIintHandler

Replaces the current interrupt handler for the specified VXIbus interrupt levels with a specified handler.



levels is a bit vector of VXI interrupt levels. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

- 1: Set.
- 0: Do not set handler.



func is the pointer to the new VXI interrupt handler.

- 0: Set to DefaultVXIintHandler.
- 3: LabVIEW occurrence handler.

I16**status.**

- 0: Successful.
 - 1: No hardware support.
-

VXlintAcknowledgeMode

Specifies whether to handle the VXI interrupt acknowledge cycle for the specified controller (embedded or extended) for the specified levels as Release on Acknowledge (ROAK) interrupts or as Release on Register Access (RORA) interrupts.



If the VXI interrupt level is handled as a RORA VXI interrupt, the local interrupt generation is automatically inhibited when the VXI interrupt acknowledge is performed. The EnableVXIint or EnableVXItoSignalInt VIs must be called to re-enable the appropriate VXI interrupt level whenever a RORA VXI interrupt occurs.

I16

controller specifies the controller (embedded or extended) for which to specify the routing.

U16

modes is a vector of VXI interrupt levels to set to RORA/ROAK interrupt mode.

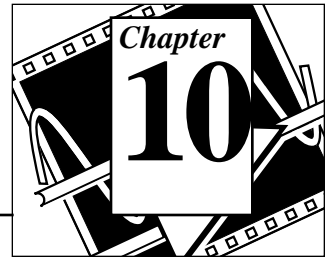
- 1: Set to RORA VXI interrupt for corresponding level.
- 0: Set to ROAK VXI interrupt for corresponding level.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

I16**status.**

- 0: VXI interrupt successfully enabled.
 - 1: No hardware support.
 - 2: Invalid **controller**.
 - 5: Invalid **modes** specified.
-

VXI Trigger VIs



This chapter describes the VIs that control triggers, a backplane feature that VXI adds to the VME standard. You can use four basic protocols (SYNC, ASYNC, SEMI-SYNC, and START/STOP) for device synchronization, for stepping through tests, or for a command path. The Trigger VIs fall into four categories:

- Source Trigger VIs act as a standard interface for asserting (sourcing) TTL and ECL triggers, as well as for detecting acknowledgements from accepting devices.
- Acceptor Trigger VIs act as a standard interface for sensing (accepting) TTL and ECL triggers, as well as for sending acknowledgements back to the sourcing device.
- Map Trigger VIs act as configuration tools for multiframe and local support for VXI triggers.
- Trigger Configuration VIs configure not only the general settings of the trigger inputs and outputs, but also the TIC counter and tick timers.

The actual capabilities of specific systems are based on the triggering capabilities of the hardware devices involved (both the sourcing and accepting devices).

Locating VXI Trigger VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»VXI Trigger** to locate the VXI Trigger VIs in LabVIEW.

Finding Help Online for VXI Trigger VIs

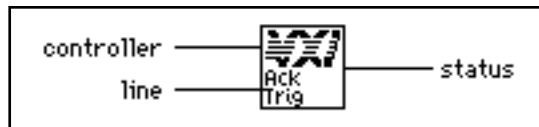
You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

VXI Trigger VI Descriptions

AcknowledgeTrig

Acknowledges the specified TTL/ECL or external (GPIO) trigger on the specified controller.



The TTL/ECL trigger interrupt handler is called after an TTL/ECL trigger is sensed. If the sensed protocol requires an acknowledge (ASync or SEMI-Sync protocols), the application should call the AcknowledgeTrig VI after performing any device-dependent operations. If you configured a trigger line using the TrigAssertConfig VI to participate in external (GPIO) SEMI-Sync acknowledging, you can use the AcknowledgeTrig VI to manually acknowledge a pending external SEMI-Sync trigger.



controller is the controller on which to acknowledge the trigger interrupt.



line is the TTL, ECL, or external trigger line to acknowledge. See the following table for the meaning of the values.

Bit	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination (GPIO 0 to 9)

I16**status.**

- 1: Successful, no need to acknowledge.
 - 0: Successful.
 - 1: Unsupportable VI (no hardware support).
 - 2: Invalid **controller**.
 - 3: Invalid **line**.
 - 4: **line** not supported.
 - 12: **line** not configured for sensing.
 - 17: No trigger sensed.
 - 18: **line** not configured for external SEMI-SYNC.
-

DisableTrigSense

Disables the sensing of the specified TTL/ECL trigger line, counter, or tick timer that was enabled by the EnableTrigSense VI.

**I16**

controller is the controller on which to disable sensing.

U16

line is the trigger line to disable sensing.

Bit	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*
60	TIC TICK timers*

*Only with controllers that have the TIC ASIC (Application Specific Integrated Circuit).



status.

- 0: Successful.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **line**.
- 4: **line** not supported.
- 12: **line** not configured for sensing.

EnableTrigSense

Enables the sensing of the specified TTL/ECL trigger line or starts up the counter or tick timer for the specified protocol.



When the protocol is sensed, the corresponding trigger interrupt handler will be invoked. In order to start up the counter or tick timers, you must first call either the TrigCntConfig or TrigTickConfig VIs, respectively.



controller is the controller on which to enable sensing.



line is the trigger line to enable sensing. See the following table for the meaning of the values.

Bit	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*
60	TIC TICK timers*

*Only with controllers that have the TIC ASIC.

U16**prot** specifies the protocol to use.

- 2: START.
- 3: STOP.
- 4: SYNC.
- 5: SEMI-SYNC.
- 6: ASYNC.

I16**status.**

- 0: Successful.
- 1: Unsupported VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **line** or **prot**.
- 4: **line** not supported.
- 5: **prot** not supported.
- 7: **line** already in use.
- 12: **line** not configured for sensing.
- 15: Previous operation incomplete.

GetTrigHandler

Returns the address of the current TTL/ECL trigger, counter, or tick timer interrupt handler for a specified trigger source.

**U16**

line is the TTL, ECL trigger line or counter/tick. See the following table for the meaning of the values.

Bit	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*

Bit	Trigger Lines
60	TIC TICK timers*

*Only with controllers that have the TIC ASIC.

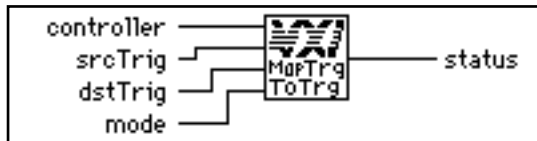
U32

func is a pointer to the current trigger interrupt handler for a specified trigger line to be used with the SetTrigHandler VI.

0: Invalid **line** or no hardware support.

MapTrigToTrig

Maps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line to another.



The support actually present is completely hardware dependent and is reflected in the error status and in hardware-specific documentation.

I16

controller is the controller on which to map signal lines.

U16

srcTrig is the source line to map to destination line.

U16

dstTrig is the destination line to map from source line. See the following table for the meaning of the values. (Star X and Star Y are not currently supported lines.)

Bit	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12

Bit	Source or Destination
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination (GPIO 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent GPIO 4 to 9
50	TIC counter pulse output (TCNTR)*
51	TIC counter finished output (GCNTR)*
60	TIC TICK1 tick timer output*
61	TIC TICK2 tick timer output*

*Only with controllers that have the TIC ASIC.



mode is the signal conditioning mode.

0: No conditioning.

Bits 0 through 3 have the following conditioning effects.

Bit	Conditioning Effect
0	Synchronize with next CLK edge.*
1	Invert signal polarity
2	Pulse stretch to one CLK minimum.*
3	Use EXTCLK (not CLK10) for conditioning

*Only with controllers that have the TIC ASIC.

All other values are reserved for future expansion.

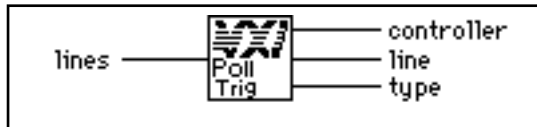


status.

- 0: Line mapped successfully.
- 1: Unsupportable VI (no mapping capability).
- 2: Invalid **controller**.
- 8: **srcTrig** not supported.
- 9: **destTrig** not supported.
- 10: **mode** not supported.
- 11: Already mapped, must use UnMapTrigToTrig.

PollTrigHandler

Waits until a trigger interrupt occurs and then continues.



lines is a bit vector that specifies which lines to poll.

- 1: Set.
- 0: Do not set.

Bit	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
14	TIC counter
15	TIC TICK timers



controller is the controller from which the trigger interrupt is received.

U16

line is the trigger line on which the trigger interrupt was received. See the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
50	TIC counter
60	TIC TICK timers

U16

type is a bit vector that specifies the type of trigger interrupt.

Bit	Trigger Lines
0	1: Sourced trigger acknowledged. 0: Trigger sensed.
2	1: Assertion edge overrun occurred
3	1: Unassertion edge overrun occurred.
4	1: Pulse stretch overrun occurred.
15	1: Error summary (2,3,4:1).



Note: *If you need timeout capabilities with this VI, use the `PollTrigHandlerTmo VI` located in Chapter 13, *VXIbus Supplemental VIs*.*

SetTrigHandler

Replaces the current TTL/ECL trigger, counter, or tick timer interrupt handler for a specified trigger source with the specified function, **func**.



U16

lines is a bit vector of trigger lines.

- 1: Set.
- 0: Do not set.

Value	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
14	TIC counter*
15	TIC TICK timers*

*Only with controllers that have the TIC ASIC.

U32

func is a pointer to the new trigger interrupt handler returned from the GetTrigHandler VI.

- 0: DefaultTrigHandler. (Calls AcknowledgeTrig if the interrupt is received while enabled to sense a trigger line using EnableTrigSense.)
- 1: DefaultTrigHandler2. (Does not call AcknowledgeTrig. The user is responsible for calling AcknowledgeTrig.)
- 3: LabVIEW Occurrence Handler.

I16

status.

- 0: Successful.
- 1: No hardware support.

SrcTrig

Sources the specified protocol on the specified TTL, ECL, or external trigger line on the specified controller.



I16

controller is the controller on which to source the trigger line.

U16

line is the trigger line to source. See the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
40 to 49	External source/destination (GPIO 0 to 9)
50	TIC counter*
60	TIC TICK timers

*Only with controllers that have the TIC ASIC.

U16

prot specifies the protocol to use.

- 0: ON.
- 1: OFF.
- 2: START.
- 3: STOP.
- 4: SYNC.
- 5: SEMI-SYNC.
- 6: ASYNC.
- 7: SEMI-SYNC and wait for acknowledge.
- 8: ASYNC and wait for acknowledge.
- FFFFH: Abort previous acknowledge pending (5 and 6).

I32

timeout specifies the timeout value in milliseconds.

I16**status.**

- 0: Trigger line sourced successfully.
 - 1: Unsupported VI (no hardware support).
 - 2: Invalid **controller**.
 - 3: Invalid **line** or **prot**.
 - 4: **line** not supported.
 - 5: **prot** not supported.
 - 6: Timeout occurred waiting for Acknowledge.
 - 7: **line** already in use.
 - 12: **line** not configured for use in sourcing.
 - 15: Previous operation incomplete.
 - 16: Previous acknowledge still pending.
-

TrigAssertConfig

Configures the specified TTL/ECL trigger line assertion method. TTL/ECL triggers can be (re-) synchronized to CLK10 on a per line basis. You can globally select all TTL/ECL trigger lines to synchronize to either the rising or falling edge of CLK10. In addition, you can specify a trigger line specified to partake in SEMI-SYNC accepting with external acknowledge.

**I16**

controller is the controller on which to configure assertion mode.

U16

line is the trigger line to configure. See the following table for the meaning of the values.

Bit	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
FFFFH	General assertion configuration (all lines).

U16

mode specifies the configuration mode.

Bit	Specific Line Configuration Modes
0	1: Synchronize falling edge of CLK10. 0: Synchronize rising edge of CLK10.

Bit	General Configuration Modes
1	1: Pass trigger through asynchronously. 0: Synchronize with next CLK10 edge.
2	1: Participate in SEMI-SYNC with external trigger acknowledge protocol. 0: Do not participate.

All other values are reserved for future expansion.

I16

status.

- 0: Successful.
- 1: Unsupported VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **line**.
- 4: **line** not supported.
- 10: Invalid configuration mode.

TrigCntrConfig

Configures TIC chip internal 16-bit counter. Call SrcTrig or EnableTrigSense to actually start the counter. The input can be any trigger line, CLK10, or the EXTCLK connection. This VI works only with controllers that have the TIC ASIC.



The counter has two outputs: TCNTR (one 100-nsec pulse per input edge) and GCNTR (unasserted until count goes from 1 to 0, then asserted until counter reloaded or reset). Use the MapTrigToTrig VI to map TCNTR to any number of the TTL or ECL trigger lines and to map GCNTR to any number of the external (GPIO) lines.

I16 **controller** is the controller on which to configure the TIC counter.

U16 **mode** is the configuration mode.

- 0: Initialize the counter.
- 2: Reload the counter leaving enabled.
- 3: Disable/abort any count in progress

U16 **source** is the trigger line to configure as input to counter.

Value	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
70	CLK10
71	EXTCLK connection

U16 **count** specifies the number of input pulses to count before terminating.

I16 **status**.

- 0: TIC counter configured successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **line** or **prot**.
- 10: Invalid configuration mode.
- 12: Counter not initialized.
- 15: Previous count incomplete.

TrigExtConfig

Configures the external trigger (GPIO) lines. You can feed back the external trigger lines for use in the crosspoint switch output. You can assert the external trigger lines high or low, or leave them unconfigured (tristated) for use as a crosspoint switch input. If the external trigger lines are not feedback, you can invert the external input before mapping it to a trigger line.


T16

controller is the controller on which to configure the external connection.

U16

extline is the trigger line to configure. See the following table for the meaning of the values.

Value	Trigger Lines
40 to 49	External source/destination (GPIO 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	EXTCLK
44 to 49	Hardware-dependent GPIO 4 to 9

U16

mode specifies the configuration mode.

Bit	Configuration Modes
0	1: Feed back any line mapped as input into the crosspoint switch. 0: Drive input to external (GPIO) pin.
1	1: Assert input (regardless of feedback). 0: Leave input unconfigured.

Bit	Configuration Modes
2	1: If assertion selected, assert low. 0: If assertion selected, assert high.
3	1: Invert external input (not feedback). 0: Pass external input unchanged.

All other values are reserved for future expansion.

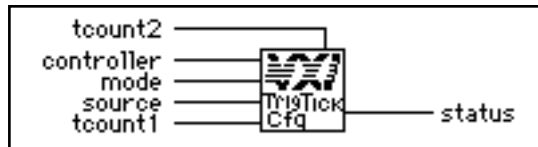
T16

status.

- 0: GPIO lines configured successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **extline**.
- 10: Invalid configuration mode.

TrigTickConfig

Configures TIC chip internal dual 5-bit tick timers. Call the SrcTrig or EnableTrigSense VIs to actually start the tick timers. The SrcTrig VI inhibits the TICK1 output from generating tick timer interrupts. The EnableTrigSense VI enables the TICK1 output to generate tick timer interrupts. The input can be any external (GPIO) line, CLK10, or the EXTCLK connection. You can map the two tick timer outputs, TICK1 and TICK2, to any number of TTL/ECL trigger lines. In addition, you can map the TICK2 output to any number of external (GPIO) lines. This VI works only with controllers that have the TIC ASIC.



U16

tcount2 is the number of input pulses (as a power of two) to count before asserting TICK2 output.

T16

controller is the controller on which to configure the TIC chip dual 5-bit tick timers.

U16

mode is the configuration mode.

- 0: Initialize the tick timers (rollover mode).
- 1: Initialize the tick timers (non-rollover mode).
- 2: Reload the tick timers leaving enabled.
- 3: Disable/abort any count in progress.

U16

source is the trigger line to configure as input to counter. See the following table for the meaning of the values.

Bit	Configuration Modes
40 to 49	External source/destination (GPIO 0 to 9)
40	Front panel in (connector 1)
41	Front panel out (connector 2)
70	CLK10
71	EXTCLK connection

U16

tcount1 is the number of input pulses (as a power of two) to count before asserting TICK1 output (and terminating the tick timer if configured for non-rollover mode).

I16

status.

- 3: Tick timers disabled successfully.
- 2: Tick timers reloaded successfully.
- 1: Non-rollover mode successfully initialized.
- 0: Rollover mode successfully initialized.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **controller**.
- 3: Invalid **source**.
- 10: Invalid configuration mode.
- 13: Invalid tcount1 or tcount2.
- 15: Previous tick configured and enabled.

UnMapTrigToTrig

Unmaps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line that was mapped to another line using the MapTrigToTrig VI.



I16

controller is the controller on which to unmap signal lines.

U16

srcTrig is the source line to unmap from destination line.

U16

dstTrig is the destination line mapped from source line. See the following table for the meaning of the values.

Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination (GPIO 0 to 9)
40	Front panel In (connector 1)
41	Front panel Out (connector 2)
42	ECL bypass from front panel
43	Connection to EXTCLK input pin
44 to 49	Hardware-dependent GPIOs 4 to 9
50	TIC counter pulse output (TCNTR)*
51	TIC counter finished output (GCNTR)*
60	TIC TICK1 tick timer output*

Value	Source or Destination
61	TIC TICK2 tick timer output*

*Only with controllers that have the TIC ASIC.

I16

status.

- 0: Line unmapped successfully.
- 1: Unsupportable VI (no mapping capability).
- 2: Invalid **controller**.
- 12: Not previously mapped.

WaitForTrig

Waits for the specified trigger line to be sensed on the specified controller for the specified time. The EnableTrigSense VI must be called to sensitize the hardware to the particular trigger protocol to be sensed.



I16

controller is the controller on which to wait for trigger.

U16

line is the trigger line to wait on. See the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*
60	TIC TICK1 tick timer*

*Only with controllers that have the TIC ASIC.

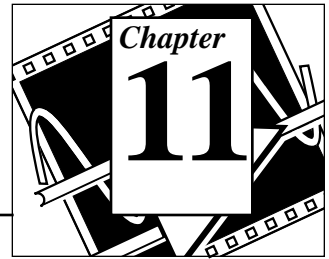
I32

timeout specifies the timeout value (in milliseconds).

I16

status.

- 0: Successful.
 - 1: Unsupportable VI (no hardware support).
 - 2: Invalid **controller**.
 - 3: Invalid **line**.
 - 4: **line** not supported.
 - 6: Timeout occurred.
 - 12: **line** not configured for sensing.
-



System Interrupt Handler VIs

This chapter describes the System Interrupt Handler VIs. You can use these VIs to handle miscellaneous system conditions that can occur in the VXI environment, such as Sysfail, ACfail, BusError, Sysreset, or Soft Reset conditions. The NI-VXI software interface can handle these system conditions for the application through the use of default interrupt service routines.

The NI-VXI software handles all system interrupt handlers in the same manner. Each type of interrupt has its own specified default handler, which is installed when InitVXIlibrary initializes the NI-VXI software. All system interrupt handlers are initially disabled (except for BusError). The corresponding enable function for each handler must be called in order to invoke the default handler.

Locating System Interrupt Handler VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»System Interrupt Handler** to locate the System Interrupt Handler VIs in LabVIEW.

Finding Help Online for System Interrupt Handler VIs

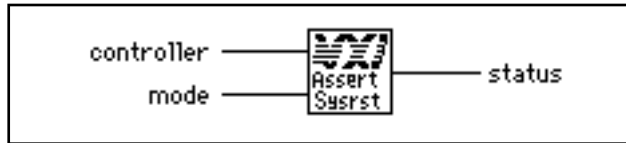
You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

System Interrupt Handler VI Descriptions

AssertSysreset

Asserts the SYSRESET* signal in the mainframe specified by **controller**.



T16

controller specifies the logical address of the mainframe extender on which to assert SYSRESET*.

- 1: From the local CPU or first extended controller.
- 2: All extenders.

U16

mode specifies the mode of execution.

- 0: Do not disturb original configuration.
- 1: Force link between SYSRESET* and local reset (SYSRESET* resets local CPU).
- 2: Break link between SYSRESET* and local reset (SYSRESET* does *not* reset local CPU).

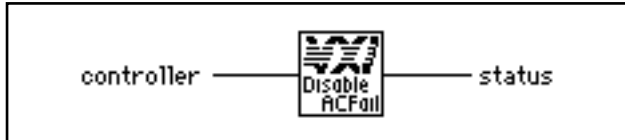
T16

status.

- 0: SYSRESET* signal successfully asserted.
 - 1: AssertSysreset not supported.
 - 2: Invalid **controller**.
-

DisableACfail

Desensitizes the local CPU from interrupts generated from ACfail conditions on the embedded CPU's VXIbus backplane, or from the specified extended controller's VXI backplane (if external CPU).



I16

controller specifies the logical address of the mainframe extender to disable.

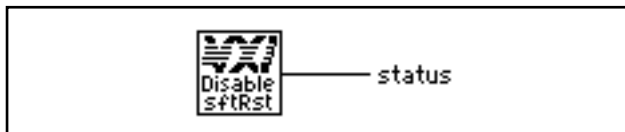
I16

status.

- 0: ACfail interrupt successfully disabled.
- 1: ACfail interrupts not supported.
- 2: Invalid **controller**.

DisableSoftReset

Disables the local Soft Reset interrupt being generated from a write to the reset bit of the local CPU control register.



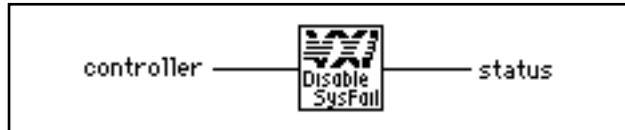
I16

status.

- 0: Soft Reset interrupt successfully disabled.
- 1: Soft Reset interrupts not supported.

DisableSysfail

Desensitizes the local CPU from interrupts generated from Sysfail conditions on the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).



I16

controller specifies the logical address of the mainframe extender to disable.

I16

status.

- 0: Sysfail interrupt successfully disabled.
- 1: Sysfail interrupts not supported.
- 2: Invalid **controller**.

DisableSysreset

Desensitizes the application from Sysreset interrupts generated from the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).



I16

controller specifies the logical address of the mainframe extender to disable.

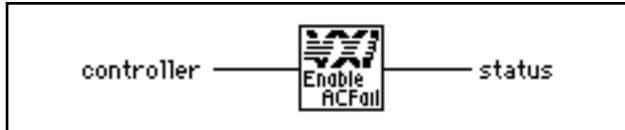
I16

status.

- 0: Sysreset interrupt successfully disabled.
- 1: Sysreset interrupts not supported.
- 2: Invalid **controller**.

EnableACfail

Sensitizes the local CPU to interrupts generated from ACfail conditions on the embedded CPU's VXIbus backplane or from the specified controller's VXI backplane (if external CPU).



I16

controller specifies the logical address of the mainframe extender to enable.

I16

status.

- 0: ACfail interrupt successfully enabled.
 - 1: ACfail interrupts not supported.
 - 2: Invalid **controller**.
-

EnableSoftReset

Enables the local Soft Reset interrupt being generated from a write to the Reset bit of the local CPU Control register.



I16

status.

- 0: Soft Reset interrupt successfully enabled.
 - 1: Soft Reset interrupts not supported.
-

EnableSysfail

Sensitizes the local CPU to interrupts generated from Sysfail conditions on the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).



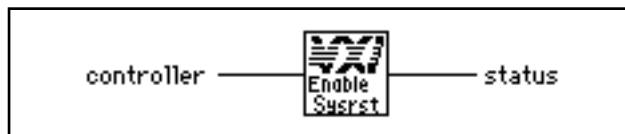
I16 **controller** specifies the logical address of the mainframe extender to enable.

I16 **status.**

- 0: Sysfail interrupt successfully enabled.
 - 1: Sysfail interrupts not supported.
 - 2: Invalid **controller**.
-

EnableSysreset

Sensitizes the local CPU to Sysreset interrupts generated from the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).



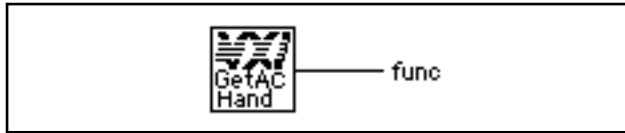
I16 **controller** specifies the logical address of the mainframe extender to enable.

I16 **status.**

- 0: Sysreset interrupt successfully enabled.
 - 1: Sysreset interrupts not supported.
 - 2: Invalid **controller**.
-

GetACfailHandler

Returns the address of the current ACfail interrupt handler.



func is the pointer to the current ACfail interrupt handler.

0: ACfail interrupts not supported.

GetBusErrorHandler

Returns the address of the current user Bus Error interrupt handler.



func is the pointer to the current Bus Error interrupt handler.

GetSoftResetHandler

Returns the address of the current Soft Reset interrupt handler.



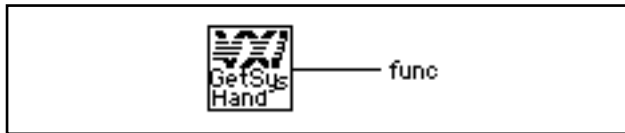


func is the pointer to the current Soft Reset interrupt handler.

0: Soft Reset interrupts not supported.

GetSysfailHandler

Returns the address of the current Sysfail interrupt handler.



func is a pointer to the current Sysfail interrupt handler.

0: Sysfail interrupts not supported.

GetSysresetHandler

Returns the address of the current SYSRESET* interrupt handler.

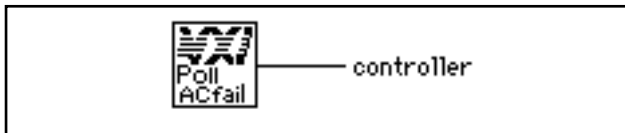


func is the pointer to the current SYSRESET* interrupt handler.

0: SYSRESET* interrupts not supported.

PollACfailHandler

Waits until an ACfail interrupt occurs and then continues.



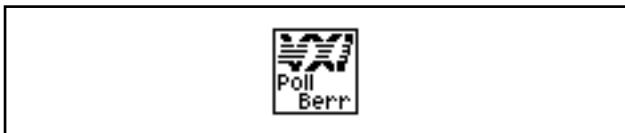
controller specifies the logical address of controller interrupting.



Note: *If you need timeout capabilities with this VI, use the **PollACfailHandlerTmo** VI located in Chapter 13, VXIbus Supplemental VIs.*

PollBusErrorHandler

Waits until a bus error occurs and then continues.



Note: *If you need timeout capabilities with this VI, use the **PollBusErrorHandlerTmo** VI located in Chapter 13, VXIbus Supplemental VIs.*

PollSoftResetHandler

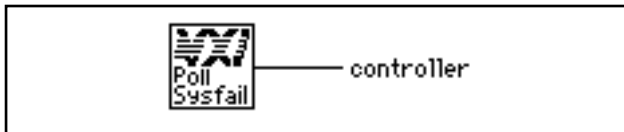
Waits until a Soft Reset interrupt occurs and then continues.



Note: *If you need timeout capabilities with this VI, use the `PollSoftResetHandlerTmo` VI located in Chapter 13, *VXIbus Supplemental VIs*.*

PollSysfailHandler

Waits until a Sysfail interrupt occurs and then continues.



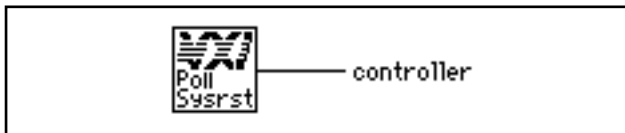
controller specifies the logical address of controller interrupting.



Note: *If you need timeout capabilities with this VI, use the `PollSysfailHandlerTmo` VI located in Chapter 13, *VXIbus Supplemental VIs*.*

PollSysresetHandler

Waits until a SYSRESET* interrupt occurs and then continues.



I16 **controller** specifies the logical address of controller interrupting.



Note: *If you need timeout capabilities with this VI, use the PollSysresetHandlerTmo VI located in Chapter 13, VXIbus Supplemental VIs.*

SetACfailHandler

Replaces the current ACfail interrupt handler with a specified handler.



U32 **func** points to the new ACfail interrupt handler.

- 0: Set to DefaultACfailHandler.
- 3: LabVIEW occurrence handler.



I16 **status.**

- 0: Successful.
- 1: ACfail interrupt not supported.

SetBusErrorHandler

Replaces the current Bus Error handler with a specified handler.



U32

func points to the new Bus Error interrupt handler.

- 0: Set to DefaultBusErrorHandler.
- 3: LabVIEW occurrence handler.

I16

status.

- 0: Successful.

SetSoftResetHandler

Replaces the current Soft Reset interrupt handler with a specified handler.



U32

func points to the new Soft Reset interrupt handler.

- 0: Set to DefaultSoftResetHandler.
- 3: LabVIEW occurrence handler.

I16

status.

- 0: Successful.
- 1: Soft Reset interrupts not supported.

SetSysfailHandler

Replaces the current Sysfail interrupt handler with a specified function, **func**.



U32

func is a pointer to the new Sysfail handler

- 0: DefaultSysfailHandler sets only the Sysfail Inhibit bit in the Control register of the failed Servant.
- 1: DefaultSysfailHandler2 sets the Reset bit along with the Sysfail Inhibit bit in the Control register of the failed Servant.
- 3: LabVIEW occurrence handler.

I16

status.

- 0: Successful.
- 1: Sysfail interrupts not supported.

SetSysresetHandler

Replaces the current SYSRESET* interrupt handler with a specified handler.



U32

func is the pointer to the new SYSRESET* interrupt handler

- 0: Set to DefaultSysresetHandler.
- 3: LabVIEW occurrence handler.



status.

- 0: Successful.
 - 1: SYSRESET* interrupts not supported.
-



VXIbus Extender VIs

This chapter describes the VXIbus Extender VIs. The NI-VXI software interface fully supports the standard VXIbus extension method presented in the *VXIbus Mainframe Extender Specification*. When the National Instruments Resource Manager completes its configuration, all default transparent extensions are complete. You can use these VIs to dynamically change these extensions if your application has such a requirement.

The transparent extensions include extensions of VXI Interrupt, TTL trigger, ECL trigger, Sysfail, ACfail, and Sysreset VXIbus signals. You can use these VIs to dynamically change these extensions if your application has such requirements. Usually, the application will never need to change the default settings. Consult your utilities manual on how to use `vxiedit`, the NI-VXI resource program editor, to change the default extender settings.

Locating VXIbus Extender VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»VXIbus Extender** to locate the VXIbus Extender VIs in LabVIEW.

Finding Help Online for VXIbus Extender VIs

You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

VXIbus Extender VI Descriptions

MapECLtrig

Maps the specified ECL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).



I16

extender specifies the mainframe extender for which to map ECL lines.

U16

lines is a bit vector of ECL trigger lines.

- 1: Enable for appropriate line.
- 0: Disable for appropriate line.

Bits 5 to 0 correspond to ECL lines 5 to 0, respectively.

U16

directions is a bit vector of directions for ECL trigger lines.

- 1: Into the mainframe.
- 0: Out of the mainframe.

Bits 5 to 0 correspond to ECL lines 5 to 0, respectively.

I16

status.

- 0: ECL trigger lines mapped successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **extender**.

MapTTLtrig

Maps the specified TTL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).



I16

extender specifies the mainframe extender for which to map TTL lines.

U16

lines is a bit vector of TTL trigger lines.

- 1: Enable for appropriate line.
- 0: Disable for appropriate line.

Bits 7 to 0 correspond to TTL lines 7 to 0, respectively.

U16

directions is a bit vector of directions for TTL trigger lines.

- 1: Into the mainframe.
- 0: Out of the mainframe.

Bits 7 to 0 correspond to TTL lines 7 to 0, respectively.

I16

status.

- 0: TTL trigger lines mapped successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **extender**.

MapUtilBus

Maps the specified VXI utility bus signal for the specified mainframe into and out of the mainframe. The utility bus signals include Sysfail, ACfail, and SYSRESET*.



extender specifies the mainframe extender for which to map utility bus signals.



modes is a bit vector of utility bus signals corresponding to the utility bus signals.

- 1: Enable for the corresponding signal and direction.
- 0: Disable for the corresponding signal and direction..

Bit	Utility Bus Signal and Direction
5	ACfail into the mainframe
4	ACfail out of the mainframe
3	Sysfail into the mainframe
2	Sysfail out of the mainframe
1	SYSRESET* into the mainframe
0	SYSRESET* out of the mainframe



status.

- 0: Utility bus signal mapped successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **extender**.

MapVXlint

Maps the specified VXI interrupt levels for the specified mainframe in the specified direction (into or out of the mainframe).



I16

extender specifies the mainframe extender for which to map VXI interrupt levels.

U16

levels is a bit vector of VXI interrupt levels.

- 1: Enable for appropriate level.
- 0: Disable for appropriate level.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

U16

directions is a bit vector of directions for VXI interrupt levels.

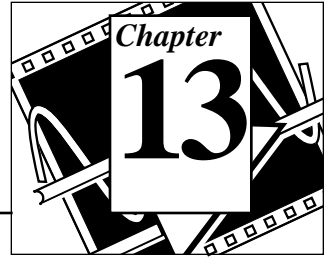
- 1: Into the mainframe.
- 0: Out of the mainframe.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

I16

status.

- 0: VXI interrupt levels mapped successfully.
- 1: Unsupportable VI (no hardware support).
- 2: Invalid **extender**.



VXIbus Supplemental VIs

This chapter describes the VXIbus Supplemental VIs. These VIs duplicate some of the previously discussed LabVIEW VIs that wait for specific interrupt conditions. However, the VXIbus Supplemental VIs have a **timeout** input, which allows you to stop waiting if the interrupt condition for which you are waiting does not occur. The polling VIs discussed in previous chapters *must* be manually stopped if the interrupt does not occur.



Note: *For all the VIs in this chapter, the timeout input defaults to infinite, which means that if you do not wire the timeout input, the VI behaves similarly to the polling VIs discussed in earlier chapters.*

Locating VXIbus Supplemental VIs in LabVIEW

Select **Windows»Show Diagram** to go to the block diagram in LabVIEW. From the **Functions** palette, choose **Instrument I/O»VXI»VXIbus Supplemental** to locate the VXIbus Supplemental VIs in LabVIEW.

Finding Help Online for VXIbus Supplemental VIs

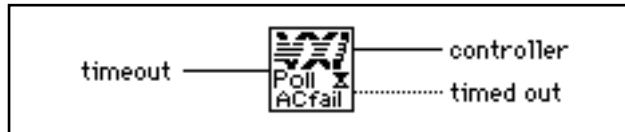
You can find much helpful information about individual VIs online by using the LabVIEW Help window. You can find the Help window by choosing **Help»Show Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click on the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

VXIbus Supplemental VI Descriptions

PollACfailHandlerTmo

Waits until an ACfail interrupt occurs.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



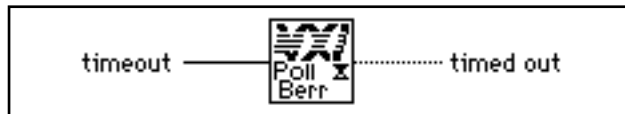
controller specifies the logical address of controller interrupting.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollBusErrorHandlerTmo

Waits until a bus error occurs.



timeout specifies the number of milliseconds to wait for the interrupt

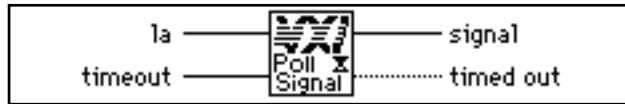
-1 Forever.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollSignalHandlerTmo

Waits until a VXI signal from the specified logical address (**la**) is received.



I16

la specifies the logical address of the signals to be handled

- 1 All known logical addresses.
- 2 Unknown (miscellaneous) signal sources.

I32

timeout specifies the number of milliseconds to wait for the interrupt

- 1 Forever.

U32

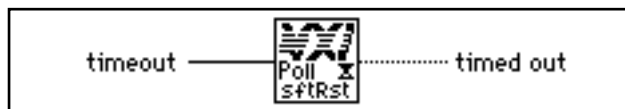
signal specifies the actual 16-bit VXI signal received.

TF

timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollSoftResetHandlerTmo

Waits until a SoftReset interrupt occurs.



I32

timeout specifies the number of milliseconds to wait for the interrupt

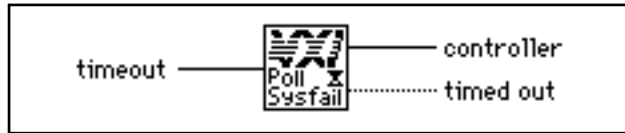
- 1 Forever.

TF

timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollSysfailHandlerTmo

Waits until a Sysfail interrupt occurs.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



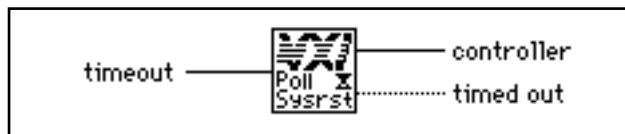
controller specifies the logical address of controller interrupting.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollSysresetHandlerTmo

Waits until a SYSRESET* interrupt occurs.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



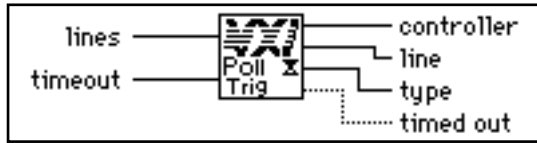
controller specifies the logical address of controller interrupting.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollTrigHandlerTmo

Waits until a trigger interrupt occurs on the specified line(s).



U16

lines is a bit vector that specifies which lines to poll.

- 1: Set.
- 0: Do not set.

Bit	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
14	TIC counter
15	TIC TICK timers

I32

timeout specifies the number of milliseconds to wait for the interrupt

- 1 Forever.

I16

controller is the controller from which the trigger interrupt is received.

U16

line is the trigger line on which the trigger interrupt was received. See the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
50	TIC counter
60	TIC TICK timers

U16

type is a bit vector that specifies the type of trigger interrupt.

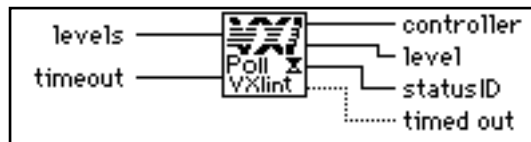
Bit	Trigger Lines
0	1: Sourced trigger acknowledged. 0: Trigger sensed.
2	1: Assertion edge overrun occurred
3	1: Unassertion edge overrun occurred.
4	1: Pulse stretch overrun occurred.
15	1: Error summary (2,3,4:1).

TF

timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollVXlintHandlerTmo

Waits until a VXI interrupt occurs on the specified level(s).



U16

levels is a bit vector that specifies which levels to poll.

- 1: set.
- 0: do not set.

Bits 0 to 6 correspond to VXI interrupt levels 1 to 7 respectively.

I32

timeout specifies the number of milliseconds to wait for the interrupt

- 1 Forever.



controller specifies the logical address of controller interrupting.



level specifies the received VXI interrupt level.



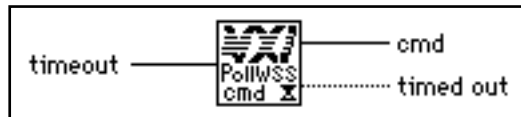
statusID specifies the status/ID obtained during IACK cycle (if it is a 16-bit VXI IACK value, it may be equivalent to a VXI signal).



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollWSScmdHandlerTmo

Waits until a Word Serial Protocol command or query is received from a VXI Message-Based Commander.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



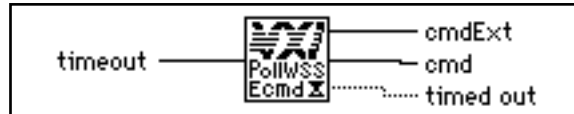
cmd is the 16-bit Word Serial command received.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollWSSecmdHandlerTmo

Waits until an Extended Longword Serial Protocol command or query is received from a VXI Message-Based Commander.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



cmdExt is the upper 16 bits of the 48-bit Extended Longword Serial command value.



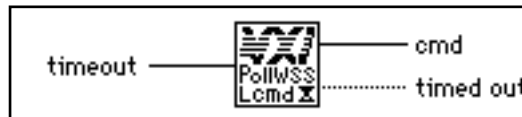
cmd is the lower 32 bits of the 48-bit Extended Longword Serial command value.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollWSSLcmdHandlerTmo

Waits until a Longword Serial Protocol command or query is received from a VXI Message-Based Commander.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



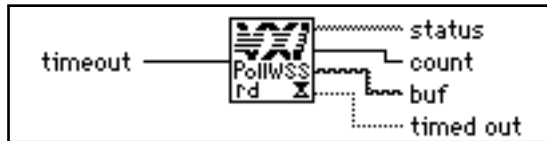
cmd is the 32-bit Longword Serial command received.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollWSSrdHandlerTmo

Waits until a Word Serial Servant read operation (started with WSSrd) terminates.



timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.



status.

The following table gives the meaning of each bit in the status value if bit 15 is TRUE.

Bit	Error Conditions (Bit 15: 1)
14	Write Ready protocol violation during transfer
13	Read Ready protocol violation during transfer
12	Data Out Ready protocol violation
11	Data In Ready protocol violation
4	WSSabort called to force abort

The following table gives the meaning of each bit in the status value if bit 15 is FALSE.

Bit	Error Conditions (Bit 15: 0)
2	All bytes received
1	END received with last byte
0	Successful transfer

U32

count is the actual number of bytes received.

abc

buf is the buffer received from WSSrd operation.

TF

timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

PollWSSwrHandlerTmo

Waits until a Word Serial Servant write operation (started with WSSwr) terminates.



I32

timeout specifies the number of milliseconds to wait for the interrupt

-1 Forever.

[TF]

status.

The following table gives the value of each bit in the status value if bit 15 is TRUE.

Bit	Error Conditions (Bit 15: 1)
14	Write Ready protocol violation during transfer
13	Read Ready protocol violation during transfer
12	Data Out Ready protocol violation
11	Data In Ready protocol violation
4	WSSabort called to force abort

The following table gives the value of each bit in the status value if bit 15 is FALSE.

Bit	Error Conditions (Bit 15: 0)
2	All bytes sent
1	END sent with last byte
0	Successful transfer

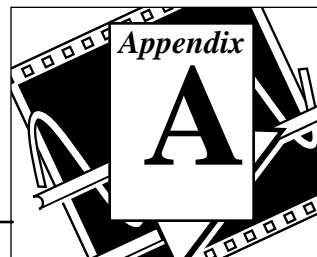


count is the actual number of bytes sent.



timed out indicates that the handler timed out. If a handler times out, the other outputs of this VI are not valid data.

Customer Communication



For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a FaxBack system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services



Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422 or (800) 327-3077

Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 1 48 65 15 59

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity



FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.



FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at the following numbers:

(512) 418-1111 or (800) 329-7177



E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate applications engineering team through e-mail at the Internet addresses listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

GPiB: gpiib.support@natinst.com

DAQ: daq.support@natinst.com

VXI: vxi.support@natinst.com

LabWindows: lw.support@natinst.com

LabVIEW: lv.support@natinst.com

HiQ: hiq.support@natinst.com

VISA: visa.support@natinst.com

Fax and Telephone Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.



Telephone



Fax

Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	519 622 9311
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 71 11
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 48301892	02 48301915
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

Fax (____) _____ Phone (____) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

List any error messages: _____

The following steps reproduce the problem: _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: LabVIEW VXI VI Reference Manual

Edition Date: November 1995

Part Number: 320557C-01

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

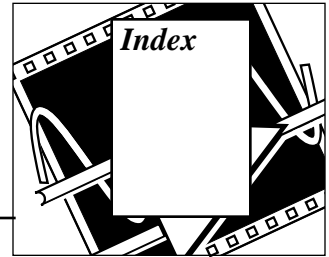
Company _____

Address _____

Phone () _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678



A

aborting Word Serial operations. *See also* WSabort VI.
Access Only Privilege, 5-3
Accepter Trigger VIs, 10-1
access privileges. *See also* High-Level VXIbus Access VIs; Low-Level VXIbus Access VIs.
 Access Only Privilege, 5-3
 Owner Privilege, 5-2 to 5-3
 same privileges and byte order for devices, 5-2
AcknowledgeTrig VI, 10-2 to 10-3
AcknowledgeVXIint VI, 9-2 to 9-3
addresses. *See also* MapVXIAddress VI; UnMapVXIAddress VI.
 for window, 5-1 to 5-2
 VXI devices, 1-1 to 1-2
AssertSysreset VI, 11-2
AssertVXIint VI, 9-3
asynchronous events and interrupts, 1-4 to 1-5

B

buffer reads, 4-2
buffer writes, 4-1 to 4-2
Byte Available Word Serial command, 4-1
byte order for devices, 5-2. *See also* GetByteOrder VI; SetByteOrder VI.
Byte Request Word Serial command, 4-2

C

Clear Word Serial command, 3-3
CloseVXIlibrary VI, 2-2
Commander. *See also* Word Serial Commander Protocol VIs.

Commander/Servant hierarchies, 1-4
 definition, 1-4
 interrupts and asynchronous events, 1-4 to 1-5
communication protocols. *See also* Word Serial Protocol.
 Commander/Servant hierarchies, 1-4
 Extended Longword Serial Protocol, 4-2
 IEEE-488 protocol, 1-3
 interrupts and asynchronous events, 1-4 to 1-5
 Longword Serial Protocol, 4-2
 software protocols (illustration), 1-3
 VXIbus Byte Transfer Protocol, 3-7, 3-12
communication registers, 1-2
configuration registers
 definition, 1-1
 illustration, 1-2
configuration VIs. *See also* System Configuration VIs.
controller parameters
 extender versus controller (illustration), 1-10
CreateDevInfo VI, 2-2
customer communication, *xvi*

D

data types, icons for, *xv*
DeAssertVXIint VI, 9-4
devices, VXI. *See also* VXI devices.
DisableACfail VI, 11-3
DisableSignalInt VI, 8-2
DisableSoftReset VI, 11-3
DisableSysfail VI, 11-4

DisableSysreset VI, 11-4
 DisableTrigSense VI, 10-3 to 10-4
 DisableVXIint VI, 9-4 to 9-5
 DisableVXItoSignalInt VI, 9-5
 documentation

- conventions used in manual, *xiv*
- icons for data types, *xv*
- organization of manual, *xiii* to *xiv*
- related documentation, *xvi*

E

embedded controllers

- embedded versus extended controllers, 1-9 to 1-10
- embedded versus external controllers, 1-8
- embedded versus external CPU configurations (illustration), 1-9

EnableACfail VI, 11-5
 EnableSignalInt VI, 8-3
 EnableSoftReset VI, 11-5
 EnableSysfail VI, 11-6
 EnableSysreset VI, 11-6
 EnableTrigSense VI, 10-4 to 10-5
 EnableVXIint VI, 9-6
 EnableVXItoSignalInt VI, 9-6 to 9-7
 errors. *See also* Word Serial Protocol errors.
 event signals.

- RouteSignal VI, 8-4 to 8-6
- SignalDeq VI, 8-7 to 8-8
- WaitForSignal VI, 8-10 to 8-12

Extended Longword Serial Protocol, 4-2
 extender parameters

- extender versus controller (illustration), 1-10

extender VIs. *See also* VXIbus Extender VIs.

F

FindDevLA VI, 2-3 to 2-4

G

GenProtError VI, 4-3
 GetACfailHandler VI, 11-7
 GetBusErrorHandler VI, 11-7
 GetByteOrder VI, 5-4
 GetContext VI, 5-5
 GetDevInfoLong VI, 2-4
 GetDevInfoShort VI, 2-5 to 2-6
 GetDevInfoStr VI, 2-6
 GetMyLA VI, 7-2
 GetPrivilege VI, 5-5 to 5-6
 GetSignalHandler VI, 8-3
 GetSoftResetHandler VI, 11-7 to 11-8
 GetSysfailHandler VI, 11-8
 GetSysresetHandler VI, 11-8
 GetTrigHandler VI, 10-5 to 10-6
 GetVXIbusStatus VI, 5-6
 GetVXIbusStatusInd VI, 5-7
 GetVXIintHandler VI, 9-7
 GetWindowRange VI, 5-8
 GetWSScmdHandler VI, 4-4
 GetWSSecmdHandler VI, 4-4
 GetWSSLcmdHandler VI, 4-5
 GetWSSrdHandler VI, 4-5
 GetWSSwrtHandler VI, 4-5

H

Handler VIs. *See also* System Interrupt Handler VIs; VXI Handler VIs; Word Serial Servant Protocol VIs.
 hardware context for windows. *See also* MapVXIAddress VI.

- definition, 5-2
- obtaining with GetContext VI, 5-5
- purpose and use, 5-2
- setting with SetContext VI, 5-11

High-Level VXIbus Access VIs, 6-1 to 6-9.
See also Low-Level VXIbus Access VIs.

- locating, 6-2
- online help, 6-2
- overview, 1-7, 6-1 to 6-2
- purpose and use, 6-1 to 6-2

VXIIin, 6-2 to 6-4
 VXIIinReg, 6-4
 VXImove, 6-5 to 6-7
 VXIout, 6-8 to 6-9
 VXIoutReg, 6-9

I

IEEE-488 protocol, 1-3
 InitVXIlibrary VI, 2-7
 Interrupt Handler VIs. *See also* System
 Interrupt Handler VIs; VXI Interrupt VIs;
 Word Serial Servant Protocol VIs.
 interrupts
 execution of user handlers, 1-5 to 1-6
 interrupts and asynchronous events, 1-4
 to 1-5
 sample interrupt handler (illustration), 1-6

L

Local Resource Access VIs, 7-1 to 7-7
 GetMyLA, 7-2
 locating, 7-1
 online help, 7-1
 overview, 1-7, 7-1
 ReadMODID, 7-2
 SetMODID, 7-3
 VXIIinLR, 7-3 to 7-4
 VXImemAlloc, 7-4
 VXImemCopy, 7-5 to 7-6
 VXImemFree, 7-6
 VXIoutLR, 7-7
 logical addresses
 obtaining with GetMyLA VI, 7-2
 VXI devices, 1-1 to 1-2
 Longword Serial Protocol, 4-2
 Low-Level VXIbus Access VIs, 5-1 to 5-14.
See also High-Level VXIbus Access VIs.
 GetByteOrder, 5-4
 GetContext, 5-5
 GetPrivilege, 5-5 to 5-6
 GetVXIbusStatus, 5-6
 GetVXIbusStatusInd, 5-7

 GetWindowRange, 5-8
 locating, 5-4
 MapVXIAddress, 5-2, 5-3, 5-9 to 5-10
 online help, 5-4
 overview, 5-1 to 5-2
 SetByteOrder, 5-3, 5-10
 SetContext, 5-3, 5-11
 SetPrivilege, 5-3, 5-12
 UnMapVXIAddress, 5-3, 5-12
 VXIpeek, 5-13
 VXIpoke, 5-14
 Low-Level VXIbus Access VIs. *See also*
 High-Level VXIbus Access VIs.
 overview, 1-7

M

mainframe support, multiple, 1-8
 manual. *See also* documentation.
 Map Trigger VIs, 10-1
 MapECLtrig VI, 12-2
 MapTrigToTrig VI, 10-6 to 10-8
 MapTTLtrig VI, 12-3
 MapUtilBus VI, 12-4
 MapVXIAddress VI, 5-2, 5-3, 5-9 to 5-10
 MapVXIint VI, 12-5
 memory management VIs. *See also* Local
 Resource Access VIs.
 Message-Based devices
 definition and overview, 1-2 to 1-3
 software protocols (illustration), 1-3
 MODID lines. *See also* Read MODID VI;
 SetMODID VI.
 multiple mainframe support, 1-8
 multiple pointer access. *See also* windows.

O

one-shot events, 1-5
 Owner Privilege, 5-2 to 5-3

P

Poll Handler VIs, 1-5
 PollACfailHandler VI, 11-9
 PollACfailHandlerTmo VI, 13-2
 PollBusErrorHandler VI, 11-9
 PollBusErrorHandlerTmo VI, 13-2
 PollSignalHandler VI, 8-4
 PollSignalHandlerTmo VI, 13-3
 PollSoftResetHandler VI, 11-10
 PollSoftResetHandlerTmo VI, 13-3
 PollSysfailHandler VI, 11-10
 PollSysfailHandlerTmo VI, 13-4
 PollSysresetHandler VI, 11-11
 PollSysresetHandlerTmo VI, 13-4
 PollTrigHandler VI, 10-8 to 10-9
 PollTrigHandlerTmo VI, 13-5 to 13-6
 PollVXIintHandler VI, 9-8
 PollVXIintHandlerTmo VI, 13-6
 PollWSScmdHandler VI, 4-6
 PollWSScmdHandlerTmo VI, 13-7
 PollWSSecmdHandler VI, 4-6
 PollWSSecmdHandlerTmo VI, 13-8
 PollWSSLcmdHandler VI, 4-7
 PollWSSLcmdHandlerTmo VI, 13-8
 PollWSSrdHandler VI, 4-7 to 4-8
 PollWSSrdHandlerTmo VI, 13-9 to 13-10
 PollWSSwrthandler VI, 4-9 to 4-10
 PollWSSwrthandlerTmo VI, 13-10 to 13-11
 privileges. *See* access privileges.
 protocols. *See also* communication protocols;
 Word Serial Protocol.

R

ReadMODID VI, 7-2
 Register-Based devices, 1-2
 registers

- configuration registers, 1-1 to 1-2
- window base register, 5-1

 resource management VIs. *See also* Local
 Resource Access VIs.
 Resource Manager (RM), 1-2

response signals

- RouteSignal VI, 8-4 to 8-6
- SignalDeq VI, 8-7 to 8-8
- WaitForSignal VI, 8-10 to 8-12

 RespProtError VI, 4-10
 RouteSignal VI, 8-4 to 8-6
 RouteVXIint VI, 9-8 to 9-9

S

Servants. *See also* Word Serial Servant
 Protocol VIs.

- Commander/Servant Hierarchies, 1-4
- definition, 1-4
- interrupts and asynchronous events, 1-4

 Set Handler VIs, 1-6
 SetACfailHandler VI, 11-11
 SetBusErrorHandler VI, 11-12
 SetByteOrder VI, 5-3, 5-10
 SetContext VI, 5-3, 5-11
 SetDevInfoLong VI, 2-7 to 2-8
 SetDevInfoShort VI, 2-8 to 2-9
 SetDevoInfoStr VI, 2-10
 SetMODID VI, 7-3
 SetPrivilege VI, 5-3, 5-12
 SetSignalHandler VI, 8-6
 SetSoftResetHandler VI, 11-12
 SetSysfailHandler VI, 11-13
 SetSysresetHandler VI, 11-13 to 11-14
 SetTrigHandler VI, 10-10
 SetVXIintHandler VI, 9-9 to 9-10
 SetWSScmdHandler VI, 4-10
 SetWSSecmdHandler VI, 4-11
 SetWSSLcmdHandler VI, 4-12
 SetWSSrdHandler VI, 4-12
 SetWSSwrthandler VI, 4-13
 Signal VIs. *See also* VXI Signal VIs.
 SignalDeq VI, 8-7 to 8-8
 SignalEnq VI, 8-9
 SignalJam VI, 8-9
 signal-type (status/ID) values

- RouteSignal VI, 8-4 to 8-6
- SignalDeq VI, 8-7 to 8-8
- WaitForSignal VI, 8-10 to 8-12

- software protocols (illustration), 1-3
 - Source Trigger VIs, 10-1
 - SrcTrig VI, 10-11 to 10-12
 - status bits
 - WSchr VI, 3-3
 - WScmd VI, 3-3 to 3-4
 - WSEcmd VI, 3-5 to 3-6
 - WSLcmd VI, 3-7 to 3-8
 - WSLresp VI, 3-8 to 3-9
 - WSrd VI, 3-9 to 3-11
 - WSresp VI, 3-12 to 3-13
 - WSstrg VI, 3-13 to 3-14
 - WSwrt VI, 3-15 to 3-16
 - System Configuration VIs, 2-1 to 2-10
 - CloseVXIlibrary, 2-2
 - CreateDevInfo, 2-2
 - FindDevLA, 2-3 to 2-4
 - GetDevInfoLong, 2-4
 - GetDevInfoShort, 2-5 to 2-6
 - GetDevInfoStr, 2-6
 - InitVXIlibrary VI, 2-7
 - locating, 2-1
 - online help, 2-1
 - overview, 1-6, 2-1
 - SetDevInfoLong, 2-7 to 2-8
 - SetDevInfoShort, 2-8 to 2-9
 - SetDevoInfoStr, 2-10
 - System Interrupt Handler VIs, 11-1 to 11-14
 - AssertSysreset, 11-2
 - DisableACfail, 11-3
 - DisableSoftReset, 11-3
 - DisableSysfail, 11-4
 - DisableSysreset, 11-4
 - EnableACfail, 11-5
 - EnableSoftReset, 11-5
 - EnableSysfail, 11-6
 - EnableSysreset, 11-6
 - GetACfailHandler, 11-7
 - GetBusErrorHandler, 11-7
 - GetSoftResetHandler, 11-7 to 11-8
 - GetSysfailHandler, 11-8
 - GetSysresetHandler, 11-8
 - locating, 11-1
 - online help, 11-1
 - overview, 1-8, 11-1
 - PollACfailHandler, 11-9
 - PollBusErrorHandler, 11-9
 - PollSoftResetHandler, 11-10
 - PollSysfailHandler, 11-10
 - PollSysresetHandler, 11-11
 - SetACfailHandler, 11-11
 - SetBusErrorHandler, 11-12
 - SetSoftResetHandler, 11-12
 - SetSysfailHandler, 11-13
 - SetSysresetHandler, 11-13 to 11-14
- ## T
- Timeout VIs
 - WSgetTmo VI, 3-6
 - WSset Tmo VI, 3-13
 - TrigAssertConfig VI, 10-12 to 10-13
 - TrigCntrConfig VI, 10-13 to 10-14
 - TrigExtConfig VI, 10-15 to 10-16
 - Trigger Configuration VIs, 10-1
 - Trigger Word Serial command, 3-13
 - TrigTickConfig VI, 10-16 to 10-17
- ## U
- UnMapTrigToTrig VI, 10-18 to 10-19
 - UnMapVXIAddress VI, 5-3, 5-12
 - user handler, execution of, 1-6
- ## V
- VXI devices. *See also* System Configuration VIs.
 - access privileges and byte orders, 5-2
 - overview, 1-1
 - VXI Handler VIs, 1-5 to 1-6. *See also* System Interrupt Handler VIs; Word Serial Servant Protocol VIs.
 - definition, 1-5
 - overview, 1-5 to 1-6
 - Poll Handler VIs, 1-5
 - Set Handler VIs, 1-6

- VXI Interrupt VIs, 9-1 to 9-10
 - AcknowledgeVXIint, 9-2 to 9-3
 - AssertVXIint, 9-3
 - DeAssertVXIint, 9-4
 - DisableVXIint, 9-4 to 9-5
 - DisableVXItoSignalInt, 9-5
 - EnableVXIint, 9-6
 - EnableVXItoSignalInt, 9-6 to 9-7
 - GetVXIintHandler, 9-7
 - locating, 9-2
 - online help, 9-2
 - overview, 1-7, 9-1
 - PollVXIintHandler, 9-8
 - RouteVXIint, 9-8 to 9-9
 - SetVXIintHandler, 9-9 to 9-10
 - VXIintAcknowledgeMode, 9-10
- VXI Signal VIs, 8-1 to 8-12
 - DisableSignalInt, 8-2
 - EnableSignalInt, 8-3
 - GetSignalHandler, 8-3
 - locating, 8-2
 - online help, 8-2
 - overview, 1-7, 8-1
 - PollSignalHandler, 8-4
 - RouteSignal, 8-4 to 8-6
 - SetSignalHandler, 8-6
 - SignalDeq, 8-7 to 8-8
 - SignalEnq, 8-9
 - SignalJam, 8-9
 - WaitForSignal, 8-10 to 8-12
- VXI Trigger VIs, 10-1 to 10-20
 - Acceptor Trigger VIs, 10-1
 - AcknowledgeTrig, 10-2 to 10-3
 - DisableTrigSense, 10-3 to 10-4
 - EnableTrigSense, 10-4 to 10-5
 - GetTrigHandler, 10-5 to 10-6
 - locating, 10-1
 - Map Trigger VIs, 10-1
 - MapTrigToTrig, 10-6 to 10-8
 - online help, 10-1
 - overview, 1-7, 10-1
 - PollTrigHandler, 10-8 to 10-9
 - SetTrigHandler, 10-10
 - Source Trigger VIs, 10-1
 - SrcTrig, 10-11 to 10-12
 - TrigAssertConfig, 10-12 to 10-13
 - TrigCntrConfig, 10-13 to 10-14
 - TrigExtConfig, 10-15 to 10-16
 - Trigger Configuration VIs, 10-1
 - TrigTickConfig, 10-16 to 10-17
 - UnMapTrigToTrig, 10-18 to 10-19
 - WaitForTrig, 10-19 to 10-20
- VXIbus
 - Commander/Servant Hierarchies, 1-4
 - configuration registers, 1-1 to 1-2
 - interrupts and asynchronous events, 1-4 to 1-5
 - Message-Based devices, 1-2 to 1-3
 - overview, 1-1 to 1-5
 - Register-Based Devices, 1-2
 - VXI devices, 1-1 to 1-2
 - Word Serial Protocol, 1-3 to 1-4
- VXIbus Byte Transfer Protocol, 3-7, 3-12
- VXIbus Extender VIs, 12-1 to 12-5
 - locating, 12-1
 - MapECLtrig, 12-2
 - MapTTLtrig, 12-3
 - MapUtilBus, 12-4
 - MapVXIint, 12-5
 - online help, 12-1
 - overview, 1-8, 12-1
- VXIbus Supplemental VIs, 13-1 to 13-11
 - locating, 13-1
 - online help, 13-1
 - overview, 13-1
 - PollACfailHandlerTmo, 13-2
 - PollBUSErrorHandlerTmo, 13-2
 - PollSignalHandlerTmo, 13-3
 - PollSoftResetHandlerTmo, 13-3
 - PollSysfailHandlerTmo, 13-4
 - PollSysresetHandlerTmo, 13-4
 - PollTrigHandlerTmo, 13-5 to 13-6
 - PollVXIintHandlerTmo, 13-6
 - PollWSScmdHandlerTmo, 13-7
 - PollWSSecmdHandlerTmo, 13-8
 - PollWSSLcmdHandlerTmo, 13-8

- PollWSSrdHandlerTmo, 13-9 to 13-10
- PollWSSwrtHandlerTmo, 13-10 to 13-11
- VXIn VI, 6-2 to 6-4
- VXInLR VI, 7-3 to 7-4
- VXInReg VI, 6-4
- VXInAcknowledgeMode VI, 9-10
- VXImemAlloc VI, 7-4
- VXImemCopy VI, 7-5 to 7-6
- VXImemFree VI, 7-6
- VXImove VI, 6-5 to 6-7
- VXIout VI, 6-8 to 6-9
- VXIoutLR VI, 7-7
- VXIoutReg VI, 6-9
- VXIpeek VI, 5-13
- VXIpoke VI, 5-14

W

- WaitForSignal VI, 8-10 to 8-12
- WaitForTrig VI, 10-19 to 10-20
- window base register, 5-1
- windows. *See also* GetWindowRange VI;
 - MapVXIAddress VI;
 - UnMapVXIAddress VI.
 - address space, 5-1 to 5-2
 - definition, 5-1
 - hardware context, 5-2
 - multiple pointer access, 5-2 to 5-4
 - Access Only Privilege, 5-3
 - Owner Privilege, 5-2 to 5-3
- Word Serial Commander Protocol VIs, 3-1 to 3-16
 - locating, 3-1
 - online help, 3-1
 - overview, 1-6
 - WSabortVI, 3-2
 - WSclr, 3-3
 - WScmd, 3-3 to 3-4
 - WSEcmd, 3-5 to 3-6
 - WSgetTmo, 3-6
 - WSLcmd, 3-7 to 3-8
 - WSLresp, 3-8 to 3-9
 - WSrd, 3-9 to 3-11
 - WSresp, 3-12 to 3-13

- WSsetTmo, 3-13
- WStrg, 3-13 to 3-14
- WSwrt, 3-15 to 3-16
- Word Serial Protocol
 - buffer reads, 4-2
 - buffer writes, 4-1 to 4-2
 - Commander/Servant
 - communication, 1-4
 - definition, 1-3, 4-1
 - Extended Longword Serial Protocol, 4-2
 - interrupts and asynchronous events, 1-4 to 1-5
 - Longword Serial Protocol, 4-2
 - purpose and use, 1-3 to 1-4
 - software protocols (illustration), 1-3
- Word Serial Protocol Errors
 - generating with GenProtError, 4-3
 - responding with RespProtError, 4-10
 - types of, 4-2
- Word Serial Servant Protocol VIs, 4-1 to 4-18
 - GenProtError, 4-3
 - GetWSScmdHandler, 4-4
 - GetWSSecmdHandler, 4-4
 - GetWSSLcmdHandler, 4-5
 - GetWSSrdHandler, 4-5
 - GetWSSwrtHandler, 4-5
 - locating, 4-2
 - online help, 4-3
 - overview, 1-7
 - PollWSScmdHandler, 4-6
 - PollWSSecmdHandler, 4-6
 - PollWSSLcmdHandler, 4-7
 - PollWSSrdHandler, 4-7 to 4-8
 - PollWSSwrtHandler, 4-9 to 4-10
 - purpose and use, 4-1 to 4-2
 - RespProtError, 4-10
 - SetWSScmdHandler, 4-10
 - SetWSSecmdHandler, 4-11
 - SetWSSLcmdHandler, 4-12
 - SetWSSrdHandler, 4-12
 - SetWSSwrtHandler, 4-13
 - WSSabort, 4-13 to 4-14

Index

- WSSdisable, 4-14
- WSSenable, 4-14
- WSSLnoResp, 4-15
- WSSLsendResp, 4-15
- WSSnoResp, 4-16
- WSSrd, 4-16 to 4-17
- WSSsendResp, 4-17
- WSSwrt, 4-18
- WSabort VI, 3-2
- WSclr VI, 3-3
- WScmd VI, 3-3 to 3-4
- WSEcmd VI, 3-5 to 3-6
- WSgetTmo VI, 3-6
- WSLcmd VI, 3-7 to 3-8
- WSLresp VI, 3-8 to 3-9
- WSrd VI, 3-9 to 3-11
- WSresp VI, 3-12 to 3-13
- WSSabort VI, 4-13 to 4-14
- WSSdisable VI, 4-14
- WSSenable VI, 4-14
- WSSsetTmo VI, 3-13
- WSSLnoResp VI, 4-15
- WSSLsendResp VI, 4-15
- WSSnoResp VI, 4-16
- WSSrd VI, 4-16 to 4-17
- WSSsendResp VI, 4-17
- WSSwrt VI, 4-18
- WStrg VI, 3-13 to 3-14
- WSwrt VI, 3-15 to 3-16